Security Protocol Design and Symbolic Analysis: Hybrid Protocols, Derived Adversary Models, and Refined Equational Theories

Thèse de doctorat de l'Université Clermont Auvergne
Spécialité: Informatique

École Doctorale Science Pour l'Ingénieur (SPI)
Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS)
CNRS, UMR 6158, LIMOS, Aubière, France

Thèse soutenue le 11 Juin 2025 par

Dhekra Mahmoud

Composition du jury:

Olivier Pereira Professeur des universités, Université Catholique de Louvain	Examinateur
Ioana Boureanu Professeure des universités, Université de Surrey	Rapporteuse
Vincent Cheval Professeur associé, Université d'Oxford	Rapporteur
Cas Cremers Professeur des universités, CISPA	Examinateur
Karthikeyan Bhargavan Directeur de recherche, INRIA	Examinateur
Jannik Dreier Maître de conférences, LORIA, Université de Lorraine	Encadrant de thèse
Pascal Lafourcade Professeur des universités, LIMOS, Université Clermont Auvergne	Directeur de thèse

LIMOS

UNIVERSITÉ
Clermont
Auvergne

Résumé

La preuve de sécurité d'un protocole, bien que formellement rigoureuse dans un modèle donné, dépend entièrement des hypothèses de ce modèle. Si les capacités de l'adversaire sont sous-spécifiées, si les primitives cryptographiques sont idéalisées, ou si les propriétés de sécurité ne sont pas formalisées, la preuve peut ne pas tenir en pratique.

La première contribution fait progresser les travaux antérieurs sur l'affinement des modèles symboliques pour les primitives cryptographiques afin de mieux capturer leurs comportements. Concrètement, nous proposons des théories équationnelles plus précises pour le cryptosystème ElGamal, les signatures DSA et les preuves à divulgation nulle de connaissance ZKP. La modélisation symbolique standard de ces primitives abstrait leurs propriétés algébriques, ce qui peut conduire à des attaques non détectées dans des protocoles plus larges. De plus, nous introduisons un modèle formel des Mix-Networks à exponentiation et re-chiffrement. En combinant ces modèles avec nos théories équationnelles, nous pouvons automatiquement détecter des attaques liées à une mauvaise utilisation des Mix-Networks, non-détéctées dans les modèles symboliques précédents.

La deuxième contribution concerne l'analyse du protocole WireGuard. Nous examinons les propriétés de sécurité revendiquées par le protocole face à un adversaire capable de compromettre toutes combinaisons possibles de clés. Pour systématiser cette analyse, nous introduisons les concepts de modèle défensif minimal et de modèle d'adversaire offensif minimal. Les modèles défensifs garantissent que la violation d'une propriété de sécurité nécessite la possession de capacités atomiques spécifiques. Les modèles offensifs minimaux définissent les plus petits ensembles de capacités atomiques suffisantes pour compromettre la sécurité. Ces dérivations ont permis d'identifier une optimisation d'implémentation introduisant de nouveaux vecteurs d'attaque.

La troisième contribution présente un protocole hybride combinant WireGuard et sa version post-quantique PQ-WireGuard, conformément aux recommandations pour une transition sécurisée vers la cryptographie post-quantique. Bien qu'une analyse symbolique de PQ-WireGuard existât, nous avons détécté une non-conformité entre le modèle et les spécifications du protocole. Nous proposons des corrections des attaques par partage de clé inconnue (UKS) trouvées, et garantissons que la sécurité du protocole hybride s'appuie à la fois sur la sécurité de la version corrigée post-quantique et la sécurité de WireGuard classique. Nous avons formellement défini la sécurité d'un protocole hybride comme nécessitant à la fois des modèles défensifs minimaux dépendant de clés post-quantiques et des modèles dépendant de clés classiques. Notre travail souligne l'importance d'une analyse itérative lors de la conception, l'atteinte des objectifs de sécurité hybride ayant nécessité des affinages répétés entre modélisation et vérification.

Abstract

The security proof of a protocol, though formally rigorous within a given model, is entirely contingent on the model's assumptions. If the adversary's capabilities are underspecified, the cryptographic primitives are idealized, or the security properties are incompletely formalized, the proof may not hold in practice.

The first contribution advances prior work on refining symbolic models for cryptographic primitives to better capture their behaviors. Specifically, we propose more precise equational theories for the ElGamal cryptosystem, DSA signatures, and Zero-Knowledge Proofs. Standard symbolic modeling of these primitives disregards their algebraic properties, which may lead to missed attacks in larger protocols. Additionally, we introduce a formal model of exponentiation and re-encryption Mix-Networks. By combining these models with our equational theories, we can automatically find attacks based on the incorrect use of the Mix-Networks missed by previous symbolic models.

The second contribution involves analyzing the WireGuard protocol. We examine the protocol's claimed security properties under an adversary capable of compromising any possible key combinations. To systematize this analysis, we introduce the concepts of minimal defensive model and minimal offensive adversary model. The defensive models ensures that violating a security property requires possessing specific atomic capabilities. Minimal offensive models define the smallest sets of adversarial capabilities that break security. Theses derivations helped to identify an implementation optimization that introduces new attack vectors.

The third contribution presents a hybrid protocol combining WireGuard and Post-Quantum WireGuard, aligning with recommendations for a secure transition to post-quantum cryptography. Although a symbolic analysis of PQ-WireGuard existed, we uncover discrepancies between the model and the protocol's specifications, including previously missed Unknown Key-Share attacks. We propose fixes and ensure the hybrid protocol's security relies on both the corrected post-quantum and classical WireGuard protocols. We formally defined a hybrid protocol's security as when there exists both minimal defensive models dependent on post-quantum keys and defensive models dependent on classical keys. Our work underscores the importance of iterative analysis during design, as achieving hybrid security required repeated refinement between modeling and verification.

Remerciements – Acknowledgments

I would like to express my deepest gratitude to Pascal and Jannik, without whom this manuscript would not have come to fruition. I am truly grateful to you.

I also wish to sincerely thank the members of my jury for agreeing to participate in my thesis defense.

Special thanks go to Ioana and Vincent for their kindness in reviewing this manuscript. Thank you very much.

Contents

Chapter 1

Introduction

> Models help us simplify the complex, but we must always remember they are approximations of reality.
>
> *Donna Strickland*

The observation that mathematical proofs based on Euclidean trigonometric identities no longer hold in non-Euclidean geometries should not be surprising, as is the inability of classical mechanics to characterize quantum-scale phenomena without a quantum-theoretic formalism. These limitations arise because all analytical frameworks rely on models, each with its own assumptions, approximations, and boundaries of applicability. This same principle applies to the analysis of security protocols. A proof of security, while formally rigorous within a given model, depends entirely on the assumptions of that model. If the adversary's capabilities are underspecified, if the cryptographic primitives are idealized, or if the security properties are incompletely formalized, then the proof may not hold in practice. The informality in protocol specification amplifies this challenge. Security protocols are typically described as sequences of message exchanges intended to guarantee properties like authentication, secrecy, and integrity. Yet, this procedural description often lacks the precision needed for formal analysis. The gap between the formal model and the informal specification means that attacks can emerge, not because the formal proof was incorrect, but because the model failed to capture adversarial behaviors not covered by the model. Thus, discovering attacks on "proven secure" protocols should not be surprising. It reflects the natural evolution of adversarial capabilities, the difficulty of fully specifying security requirements, and also the inherent limitations of modeling real-world systems. The ongoing refinement of formal methods in security protocols aims to narrow this gap while acknowledging that no model can ever be perfectly complete.

To illustrate the inherent challenges in modeling security protocols and to avoid remaining overly abstract, we examine a classic case study of a security protocol from the literature. In 1978, Roger M. Needham and Michael D. Schroeder proposed the so-called *Needham-Schroeder Public-Key Protocol* intended to guarentee the mutual authentication of the communicated entities over an insecure network [NS78]. Mutual authentication, according to [NS78], simply means "*verifying the identity of the communicating principals to one another*". In order to describe a simplified version of the protocol, let us assume that public keys $\mathsf{pk}_{\mathcal{R}}$ are known and commonly associated to the users identities $\mathcal{R}$. The Needham-Schroeder protocol then works as depicted in Figure 1.1. An initiator $\mathcal{I}$ who wants to communicate with $\mathcal{R}$, generates an arbitrary number $n_{\mathcal{I}}$ used only once, i.e. a nonce, encrypts the nonce together with their identity using the public key of the intended responder $\mathcal{R}$ and sends the message $\{n_{\mathcal{I}}, \mathcal{I}\}_{\mathsf{pk}_{\mathcal{R}}}$ to $\mathcal{R}$. Upon receiving the message from $\mathcal{I}$, the receiver generates a nonce $n_{\mathcal{R}}$ and sends it back together with the nonce $n_{\mathcal{I}}$ encrypted using the public key of the initiator. $\mathcal{I}$ receives the message from $\mathcal{R}$ and if it has the correct form, $\mathcal{I}$ would have authenticated $\mathcal{R}$. The handshake is considered complete when $\mathcal{I}$ sends back the encryption of $n_{\mathcal{R}}$ to $\mathcal{R}$ so that $\mathcal{R}$ can authenticate $\mathcal{I}$.

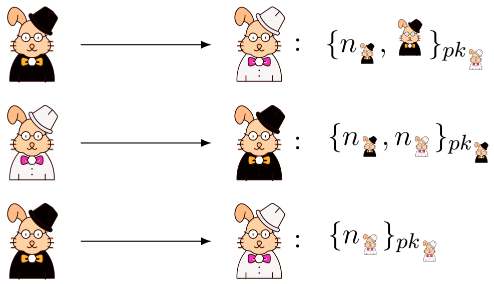


Figure 1.1: The Needham-Schroeder Public-Key Protocol.

When we described the Needham-Schroeder protocol above, we used two approaches. Figure 1.1 corresponds to the commonly used protocol *narration* in the cryptographic protocol litterature and which consist of a simple sequence of messages exchanged between the protocol's participants [BN05]. Beyond its informality, this approach may incorporate numerous implicit assumptions that can introduce ambiguities during protocol security analysis. For instance, it is not clear from Figure 1.1 how the values n and n are generated. The second description we provided is a purely textual representation of the protocol. However, human language is inherently ambiguous, making it difficult to formulate descriptions that admit only one precise interpretation. The same reasoning hold for the above definition of *authentication*. What exactly does authentication mean? If Alice runs an authentication protocol with another party, and an attacker modifies her message without changing her identity, is she still considered authenticated? It is her running the protocol and her identity appears in the message, so does the other party consider her authenticated? In [Gol96], Dieter Gollmann asks the question "*What do we mean by entity authentication?*" and observes that authentication protocol design has been particularly error-prone due to problems in defining what authentication means. A **formal language** with well-defined syntax, grammar, and semantics is necessary to describe cryptographic protocols and their **security properties** for correct analysis.

However, formal specifications of both the protocol and its security properties remain insufficient to ensure rigorous analysis. In their work [BAN90], Burrows et al. introduced a logic for modeling and analyzing authentication protocols, demonstrating that the Needham-Schroeder public-key protocol ensures mutual authentication between trustworthy communicating parties. However, several years later, Gavin Lowe [Low95] uncovered an attack on this protocol, showing that a dishonest participant could impersonate an honest user by running two sessions in parallel, i.e., two execution instances of the protocol *Key Lesson:* When analyzing protocols, the assumed power of an **adversary** plays a decisive role in the security guarantees.

A fundamental challenge exists in determining the necessary bounds on protocol sessions to ensure the absence of attacks, and the appropriate adversarial model to assume. The standard solution for session bounds is to verify protocol security under unbounded session assumptions. While executing infinite sessions is unrealistic in practice, this approach provides strong security guarantees. Crucially, it eliminates the need to define which bound is sufficient for a given protocol and property. However, security analysis under unbounded session assumptions typically results in undecidable problems [DLMS99]. Bounding the number of sessions in security analysis is also non-trivial because an attacker can insert any number of messages into chosen sessions, including messages of unlimited length and instantiating an unlimited number of nonces.

Another aspect not yet emphasized, yet apparent in the protocol description in Figure 1.1, is its cryptographic foundations. The Needham-Schroeder protocol above is based on public-key cryptography, raising other several fundamental questions: Is the encryption scheme deterministic, probabilistic, or homomorphic? Which mathematical properties does it employ? Does it use modular exponentiation or pairing operations? Furthermore, would the protocol maintain equivalent security guarantees when analyzed with different public-key algorithms? **Cryptographic primitives** inherently possess distinct algebraic and security characteristics, making it obvious that protocol instantiation with different primitives yields non-equivalent security properties. Deepak Kapur et al. demonstrated in [KNW03] that security protocol analysis becomes undecidable when considering cryptographic primitives involving exponentiation, under the assumption of exponentiation's distributivity over multiplication.

The preceding discussion reveals fundamental challenges in security protocol design, modeling, and analysis. Manual verification of such protocols consequently is both error-prone and computationally intensive when addressing this complexity. The literature abounds with mathematical proofs that were subsequently demonstrated to be either flawed or incomplete. While automated verification tools exist to assist with analysis, significant challenges persist in three key aspects: accurate protocol modeling, precise specification of security properties and rigorous definition of adversarial capabilities. Each of these components introduces substantial complexity and potential for error. This thesis fits within this context and displays subtle results found during various analyses with automated formal verification tools in the so-called symbolic model through the enrichment of already existing models.

The Symbolic Model. The symbolic model, due to Dolev and Yao [DY83], assumes *perfect cryptography:* one-way funtions are unbreakable, and one can decrypt only in possession of the secret key. While this assumption may seem overly restrictive for cryptographic protocol analysis, protocols encompass far more than just their underlying primitives. Even when using cryptographically secure algorithms, vulnerabilities can emerge at the protocol level itself. The attack on the Needham-Schroeder protocol, as documented by Lowe, serves as an illustrative case where an attacker compromise the protocol's security in the absence of any cryptographic vulnerabilities. This establishes that protocol-level flaws may exist independently of cryptographic assumptions.

In the symbolic model, messages are represented as terms which can be either atomic (to represent fresh values such as keys or random coins) or constructed by applying function symbols. For instance, the terms $\mathsf{pk}(sk)$, $\mathsf{aenc}(m, \mathsf{pk}(sk))$ and $\mathsf{adec}(c, sk)$ can represent a public key function, an asymmetric encryption and an asymmetric decryption, respectively. To model the behavior of cryptographic primitives, function symbols may be ruled by a set of equations, i.e., an *equational theory.* For example, the equation $\mathsf{adec}(\mathsf{aenc}(m, \mathsf{pk}(sk)), sk) = m$ states that decrypting an encrypted message m using the proper key pairs sk and $\mathsf{pk}(sk)$, results in the same message m. It is noteworthy to mention that the sole equalities between the terms are those explicitly specified by the equations. If no equation is added for $\mathsf{pk}(sk)$, then it behaves as a perfect one-way function. The latter equation is the standard equation used to model public key encryption in the symbolic model.

The Dolev-Yao attacker model is typically the assumed adversary in symbolic models. As mentioned before, the attacker can perform computations only according to the equational theory specified in the model. The Dolev-Yao attacker also has complete control over the network. The attacker's capabilities consist of injecting, eavesdropping on, removing, duplicating, replaying, substituting, and delaying all messages transmitted during protocol execution.

Remark:

The *computational model* [GM84] is another model for analyzing security protocols where messages are bitstring and the adversary is a probabilistic polynomial time Turing machine. Since this manuscript focuses exclusively on the symbolic model, we do not provide further details here. For comprehensive coverage of both symbolic and computational approaches, we refer readers to [CKW09].

Trace and equivalence properties. There are two main classes of security properties in the symbolic approach, namely *trace properties* and *equivalence properties.* Trace properties are defined with regard to the executions of the protocol. A trace property is considered to be satisfied when, in all possible traces or executions of the protocol, the property holds. Several security properties can be expressed as trace properties, such as secrecy (as a reachability property, that is, whether the attacker can *reach* or not a state where they know some secret terms) and authentication (as correspondence assertions, that is, whenever a specific state is reached by one protocol participant, another state must be previously reached by another participant). Most of the existing symbolic tools supports the specification of trace properties.

Equivalence properties are defined between two scenarios that the attacker should not be able to tell apart. Privacy properties can be expressed as equivalence properties, such as anonymity, unlinkability and strong secrecy. Proving that a property holds with equivalences provides a stronger guarantee than when it is proven as a trace property. For example, analyzing the secrecy for some term M when expressed as an equivalence property (M is indistinguishable from some newly generated randomness) is stronger than analyzing the secrecy of M expressed as a simple reachability property. There are two major equivalence properties supported by the existing automated protocol verifiers: *trace equivalence* and *observational equivalence.* Trace equivalence is weaker in the sense that it is implied by observational equivalence [CD09]. In [BDM20, HBD17, BDD23], the authors state that observational equivalence may be too strong for privacy properties and show how trace equivalence may be the most suitable property for the matter. However, in [HM20], the authors exhibit a linkability attack on the e-Passport protocol using the *labeled bisimilarity* property (labeled bisimilarity and observational equivalence are equivalent [ABF17]), although the protocol is proven to be unlinkable in [HBD17] using trace equivalence.

Automated Symbolic Tools. Several tools exist for verifying security protocols in the symbolic model. The applied Π-Calculus, is among the most used language for modeling security protocols and constitutes the input language of many tools. For the full details about its syntax and semantics, see Section 2.1 of Chapter 2. In the automatic cryptographic protocol verifier ProVerif, protocols are described in the applied Π-Calculus. The tool can handle many cryptographic primitives (encryption, signature, hash function, Diffie-Hellman Key agreement) specified as rewrite rules or equations, but cannot handle associativity [Bla14] preventing modeling primitives such as XOR. ProVerif can also handle an unbounded number of sessions and an unbounded message space at the cost of (potential) non-termination. It is able to prove traces properties and observational equivalence between processes that differ only by terms. ProVerif is sound but not complete which means that it can give spurious attacks or fail to conclude. The tool has been used successfully to analyze many real-world protocols such as TLS [BCW22, BBK17] and Messaging protocols [KBB17, BJKS24].

In the security protocol verification tool Tamarin, protocols are described using multiset rewrite rules. The protocols' states are defined by a number of facts and rewrite rules describing how the states evolve. The rules induce a transition system: a rule application removes some facts from the current state and adds other facts, hence transitioning to a new state. In Tamarin, the transitions are labeled with actions which are facts that annotate rules, and are used to specify properties. The tool uses first-order time-points to reason about security properties with regard to the protocol executions expressed as sequence of actions generated by the multiset rewrite rules. It handles trace properties, expressed as lemmas, and equivalences between processes with regard to an unbounded number of sessions. Tamarin supports natively associative and commutative symbols such as Diffie-Hellman exponentiation. The tool is both sound and complete in the trace mode [BCDS17].

The verification tool DeepSec [CKR24] supports only equivalence-based properties while bounding the number of sessions. Protocols are also described in the applied Π-Calculus. The equivalence property analyzed by DeepSec, is trace equivalence.

Sapic$^+$ [CJKK22] unifies the use of ProVerif, Tamarin and DeepSec. It was recently used to analyze EDHOC protocol [JKKR23]. Protocols are modeled in the applied Π-Calculus in a single Sapic$^+$ file which enables the generation of three state-of-the-art protocol models for Tamarin, ProVerif and DeepSec. Sapic$^+$ allows to exploit the strengths of each tool, which justifies our choice for this framework in Chapters 4 and 5.

Contribution and Organization of the Thesis

This thesis is situated within the research context of modeling the security protocols and their formal analysis conducted with state-of-the-art automated formal verification tools, and contributes by presenting nuanced findings uncovered through a series of rigorous analyses. Specifically, this work extends and enhances preexisting models by introducing methodological refinements and additional layers of formal verification using automated tools. By building upon established formal models, we uncovers new dimensions of analysis that were previously unexamined or insufficiently characterized. More specifically, our contributions are the following:

In Chapter 3, we propose more precise equational theories for ElGamal cryptosystem, DSA signatures and Zero Knowledge Proofs (ZKPs) for the Discrete Logarithm (DL) knowledge. The standard symbolic modeling of the latter primitives abstracts away any algebraic and homomorphic properties, thus potentially missing attacks when carried out with larger protocols. We also propose a formal model of exponentiation and re-encryption Mix-Networks (Mix-Nets) in the applied Π-Calculus. We show that using these models together with our equational theories, we can automatically discover attacks based on the incorrect use of the Mix-Nets, thus, helping protocol designers avoiding these and similar attacks in the future. In particular, we find attacks on four cryptographic protocols using ProVerif: we show that an electronic exam protocol, two electronic voting protocols, and the "Crypto Santa" protocol do not satisfy the desired privacy properties. We then fix the vulnerable protocols by adding missing ZKPs and re-analyze the resulting protocols using ProVerif. We show that in this case of a weak ZKP (malleable) all attacks persist, and that we find again these attacks automatically.

In Chapter 4, we propose a symbolic analysis of the WireGuard protocol using the Sapic$^+$ framework. Our models are compatible with both the automatic cryptographic protocol verifiers ProVerif and Tamarin using the Sapic$^+$ framework. We consider a complete protocol execution which includes cookie messages used for resistance against denial of service attacks. We model different adversarial capabilities, namely adversaries that can read or set static, ephemeral or pre-shared keys, read or set Elliptic Curve Diffie-Hellman (ECDH) pre-computations, and control public key distribution. Eventually, we present our results in a unified and interpretable way, allowing comparisons with previous analyses. Finally, thanks to our models, we give necessary and sufficient conditions for each analyzed security property to be compromised. We confirm a flaw on the anonymity of the communications and point out an implementation choice which considerably weakens the security of the protocol. We then propose remediations that we prove secure.

In Chapter 5, we first conduct a thorough formal analysis of the Post-Quantum WireGuard protocol. Thanks to our analysis, we point out and fix a number of weaknesses. This leads us to an improved construction PQ-WireGuard*. Secondly, we propose and formally analyze a new protocol, based on both WireGuard and PQ-WireGuard*, named Hybrid-WireGuard, compliant with current best practices for post-quantum transition about hybridization techniques. For our analysis, we use the SAPIC$^+$ framework that enables the generation of three state-of-the-art protocol models for the verification tools PROVERIF, DEEPSEC and TAMARIN from a single specification, leveraging the strengths of each tool. We formally prove that Hybrid-WireGuard is secure.

To make the manuscript more accessible, we introduce in Chapter 2, the formalism required to grab the introduced works.

Publications

All of the research presented in this thesis has previously been published or submitted to academic conferences. The analysis of the protocols using the MIX-NETS models of Chapter 3 was presented at the USENIX Security Symposium 2024 [DLM24]. The analysis of the WireGuard protocol described in Chapter 4 was presented at the NDSS Symposium 2024 [LMR24]. The analysis of the PQ-WireGuard and the design of the hybrid Wireguard is submitted at the USENIX Security Symposium 2025.

Below are presented the title and the abstract of the remaining works published or submitted during this thesis, yet not part of this manuscript.

- Formal Analysis of SDNsec: Attacks and Corrections for Payload, Route Integrity and Accountability [HMLP25].
    - *Abstract:* Modern networks increasingly rely on Software Defined Networking (SDN) to achieve flexibility and efficiency, which makes it more critical to ensure their security. Particularly, it is crucial to ensure the integrity of packets routing within the Network. In this paper, we address key security challenges within SDN architectures with a particular focus on the data plane. We ex- plore these challenges through the lens of a security solution called SDNsec. We identify three fundamental security properties, namely payload integrity, route integrity and accountability. We define these security properties in the formal framework of the applied Π-CALCULUS. Likewise, we

suggest two levels of route integrity: strong and weak, depending on the protocol's requirements. We use ProVerif, a cryptographic protocol verifier, to conduct a formal analysis of the security of SDNsec. Thanks to our models, we discover several flaws on all the afore- mentioned properties. In response, we propose corrective measures to enhance SDNsec's security. Moreover, to ensure that such attacks are feasible and that our improvements are effective, we implement and test SDNsec and our corrected solution using the RYU controller and Mininet network emulator.

- Transferable, Auditable and Anonymous Ticketing Protocol [LMMOA24]
  - *Abstract:* Digital ticketing systems typically offer ticket purchase, refund, validation, and, optionally, anonymity of users. However, it would be interesting for users to transfer their tickets, as is currently done with physical tickets. We propose Applause, a ticketing system allowing the purchase, refund, validation, and transfer of tickets based on trusted authority, while guaranteeing the anonymity of users, as long as the used payment method provides anonymity. To study its security, we formalise the security of the transferable E-Ticket scheme in the game-based paradigm. We prove the security of Applause computationally in the standard model and symbolically using the protocol verifier ProVerif. Applause relies on standard cryptographic primitives, rendering our construction efficient and scalable, as shown by a proof-of-concept. In order to obtain Spotlight, an auditable version, proved to be secure, users will remain anonymous except for a trusted third party, which will be able to disclose their identity in the event of a disaster.
- Formal Analysis of C-ITS PKI protocols [MLM24]
  - *Abstract:* Vehicular networking is gaining a lot of popularity and attraction from among the industry and academic research communities in the last decade. The communication between vehicles will lead to more efficient and secured roads because we will be able to provide information about traffic and road conditions to vehicle's drivers. However, ensuring the security of these networks and devices still remains a main major concern to guarantee the expected services. Secure Public Key Infrastructure (PKI) represents a common solution to achieve many security and privacy requirements. Unfortunately, current Cooperative Intelligent Transport Systems (C-ITS) PKI protocols were not verified in terms of security and privacy. In this paper, we propose a security analysis of C-ITS PKI protocols in the symbolic model

using ProVerif. We formally modeled C-ITS PKI protocols based on the specifications given in the ETSI standard. We model C-ITS PKI protocols and formalize their security properties in the applied Pi-calculus. We used an automatic privacy verifier UKano to analyse Enrolment protocol. We found attacks on authentication properties, in Authorization and Validation protocols when considering a dishonest Authorization Authority (AA).We analysed proof results and we fixed identified attacks by introducing new parameters in protocol request.

- Jannik Dreier, Rosario Giustolisi, Pascal Lafourcade, Gabriele Lenzini, Dhekra Mahmoud and Mohammadamin Rakeei. Secure and Verifiable Coercion-Resistant Electronic Exam submitted at the Journal of Cybersecurity on September 2024.

  – *Abstract:* Since they enable efficient assessment of large cohorts of students and test-takers, electronic exams (e-exams) have become popular. However, the transition from pencil-and-paper tests to e-exams comes with challenges: researchers needed to ensure a comparable level of security and privacy as that enjoyed before the transition; at the same time, they have to address threats due to the use of information and communication technology. Research has shown that, for the reason of assessment fairness, e-exams should satisfy a list of peculiar security properties, for instance, about authentication, secrecy, integrity, anonymity, and correctness, including their universal and individual verifiability. Recently, e-exams have been scrutinized for their resistance to collusion and coercion. Subsets of participants have an interest in teaming up, or forcing one another, to gain an unfair advantage over the honest others. In this work, we study coercion-resistance for e-exams. We propose a novel strong definition of coercion where all secrets are leaked to the attacker. Under this threat, we prove that a recent coercion-resistant exam protocol is subject to attacks. We improve the protocol by ensuring that all its properties are maintained and that it is coercion-resistant under the new threat model. Our new protocol is also verifiable, which is a must-have property whenever there is the need to prove that fairness is preserved despite anyone attempting to subvert it. All our claims are formally verified using ProVerif. Notably, our formal verification includes proving the security of a recent exponentiation mixnet framework proposed in the literature.

Chapter 2

Background

A proof without a
machine-checkable formalization
is just a story.

Shafi Goldwasser

Contents

In this introductory chapter, we will begin by presenting the applied Π-Calculus language, as throughout this manuscript, all the analyzed cryptographic protocols are described using this language. This applies both to Chapter 3, where we used ProVerif, and to Chapters 4 and 5, where we used Sapic^{+}. Next, we will introduce the cryptographic primitives used in the protocols we analyzed. However, we will not delve deeply into the cryptographic details themselves and will instead remain at a highly abstract level. The computational problems mentioned here are merely necessary elements to describe the protocols in a high-level manner. Moreover, since this thesis is fully situated within the context of formal verification and symbolic modeling of cryptographic protocols, we describe - for each cryptographic primitive - the common modeling approaches used to model this primitives in the symbolic model.

2.1 The Applied Π-Calculus

The applied Π-Calculus is a language for modeling and analyzing security protocols introduced in 2001 by Abadi et al. [ABF17]. It is an extension of Milner's calculus [Mil89], a mathematical formalism for describing and analyzing properties of concurrent computation. As the applied Π-Calculus serves as the formal foundation for all the analzed protocols in this thesis (encoded either directly in ProVerif or via Sapic$^+$), we begin by recalling its syntax, semantics, and equivalence relations. The following is mainly based on [Dre13, RS11].

Syntax. A finite *signature* Σ is a finite set of function symbols each with an associated arity. Given a signature Σ, an infinite set of names $\mathcal{N}$ and an infinite set of variables $\mathcal{V}$, the set of *terms* $\mathcal{T}$ is defined as names, variables and function symbols applied to other terms. The set of terms $\mathcal{T}$ is defined by the grammar in Table 2.1 where f ranges over the functions in Σ and l matches the arity of f. Terms are equipped with an equational theory (a set of equations), which induces an equivalence relation between terms, denoted $=_E$. As an illustrative example, the behavior of probabilistic asymmetric encryption and decryption can be formally modeled using the following equational theory:

$$\mathsf{dec}(\mathsf{enc}(m, \mathsf{pk}(x), r), x) =_E m \tag{2.1}$$

where pk is a unary function symbol representing the public key corresponding to the secret key x, m is the plaintext, and r represents a random variable. The sole equalities between the terms are those explicitly specified by the equational theory.

Remark:

We recall that a binary relation $\equiv$ on a set $\mathcal{S}$ is said to be an equivalence relation if it has the following properties:

- *Reflexivity*: $\forall x \in \mathcal{S}, x \equiv x$.
- *Symmetry*: $\forall x, y \in \mathcal{S}, x \equiv y \iff y \equiv x$.
- *Transitivity*: $\forall x, y$ and $z \in \mathcal{S}$, if $x \equiv y$ and $y \equiv z$, then $x \equiv z$.

Systems are described as *processes*. The grammar for processes is depicted in Table 2.2, where M and N are terms, n is a name, x is a variable and c is a channel name. The null process $\mathbf{0}$ does nothing; $P \mid Q$ is the parallel composition of P and Q; the replication $!P$ behaves as an infinite number of copies of P running in parallel. The process $\nu n \cdot P$ makes a new private name n then it runs P. Finally, $\mathtt{in}(c, x)$ and $\mathtt{out}(c, N)$ stand for an input and an output on the channel c respectively.

$L, M, N ::=$	terms
a, b, c, n	name
x, y, z	variable
$f(M_1, \ldots, M_l)$	function application

Table 2.1: Grammar of terms in the applied Π-Calculus.

$P, Q, R ::=$	
$\mathbf{0}$	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu n \cdot P$	name restriction
if $M = N$ then P else Q	conditional
$\mathtt{in}(c, x) \cdot P$	message input
$\mathtt{out}(c, N)$	message output

Table 2.2: Grammar of processes in the applied Π-Calculus.

Processes are also extended with *active substitutions.* We write $\{^M/_x\}$ for the substitution that replaces the variable x with the term M. The grammar of extended processes is given in Table 2.3. $\{^M/_x\}$ is considered as a process and $\nu x \cdot (\{^M/_x\}|P)$ corresponds to $\mathtt{let}\ x = M\ \mathtt{in}\ P$. A *frame* is defined as an extended process built from $\mathbf{0}$ and active substitutions by parallel composition and restrictions. The *domain* of a frame is the set of variables for which the frame defines a substitution, and which are not under restriction. An extended process A is *closed* when its free variables are all defined by an active substitution. A *context* $\mathcal{C}[\cdot]$ is defined as a process with a hole, that can be filled with any process. An evaluation context is a context whose hole is not under a replication, a condition, an input or an output. A context $\mathcal{C}[\cdot]$ closes A when $\mathcal{C}[A]$ is closed.

$A, B, C ::=$	
P	plain process
$A \mid B$	parallel composition
$!A$	replication
$\nu n \cdot A$	name restriction
$\nu x \cdot A$	variable restriction
$\{^M/_x\}$	active substitution

Table 2.3: Grammar of extended processes.

Semantics and Equivalences. The applied Π-Calculus' operational semantics is defined by: 1) structural equivalence ($\equiv$), 2) internal reduction ($\rightarrow$), and 3) labelled reduction ($\xrightarrow{\alpha}$).

Informally, two processes are *structurally* equivalent when they model the same behavior despite differing syntactic encodings permitted by the grammar. Formally, structural equivalence of processes is the smallest equivalence relation on extended processes that is closed by α-conversion on names and variables, by application of evaluation contexts as shown is Table 2.4. *Internal reduction* is the smallest relation on extended processes closed under structural equivalence following the rules in Table 2.5. *Labeled reduction.* models the environment interacting with the processes. It defines a relation $A \xrightarrow{\alpha} A'$ between two extended processes where α is either reading a term or sending a name or variable (See Table 2.6).

PAR - 0	$A \mid 0$	$\equiv$	A	
PAR - A	$A \mid (B \mid C)$	$\equiv$	$(A \mid B) \mid C$	
PAR - C	$A \mid B$	$\equiv$	$B \mid A$	
REPL	$!P$	$\equiv$	$P \mid !P$	
SUBST	$\{^M/_x\} \mid A$	$\equiv$	$\{^M/_x\} \mid A\{^M/_x\}$	
NEW - 0	$\nu u \cdot 0$	$\equiv$	0	
NEW - C	$\nu u \cdot \nu v \cdot A$	$\equiv$	$\nu v \cdot \nu u \cdot A$	
NEW - PAR	$A \mid \nu v \cdot B$	$\equiv$	$\nu v \cdot (A \mid B)$	if $v \notin fn(A) \cup fv(A)$
ALIAS	$\nu x \cdot \{^M/_x\}$	$\equiv$	0	
REWRITE	$\{^M/_x\}$	$\equiv$	$\{^N/_x\}$	if $M =_E N$

Table 2.4: Structural equivalence in the applied Π-Calculus.

Static Equivalence. Two terms M and N are equal in the frame B, written as $(M =_E N)B$, if and only if there exists a set of restricted names $\overline{n}$ and a substitution σ such that $B \equiv \nu \cdot \sigma$, $M\sigma =_E N\sigma$ and $\cap(fn(M) \cup fn(N)) = \varnothing$. Closed frames B and B' are statically equivalent, denoted $B \approx_s B'$, if:

1. $dom(B) = dom(B')$

COMM	$\overline{c}\langle x\rangle \cdot P \mid c(x) \cdot Q \rightarrow P \mid Q$
THEN	$if\ \ N =_E N\ then\ P\ else\ Q \rightarrow P$
ELSE	$if\ \ M =_E N\ then\ P\ else\ Q \rightarrow P$
	for ground terms M, N where $M \neq_E N$

Table 2.5: Internal reduction in the applied Π-Calculus.

IN	$\mathtt{in}(c,x)\cdot P \xrightarrow{\mathtt{in}(c,M)} P\{^{M}/_{x}\}$
OUT-ATOM	$\mathtt{out}(c,v)\cdot \xrightarrow{\mathtt{out}(c,v)} P$
OPEN-ATOM	$\dfrac{A \xrightarrow{\mathtt{out}(c,v)} A' \quad v \neq c}{\nu v \cdot A \xrightarrow{\nu v\cdot\mathtt{out}(c,v)} A'}$
SCOPE	$\dfrac{A \xrightarrow{\alpha} A' \quad v \text{ does not occur in } \alpha}{\nu v \cdot A \xrightarrow{\alpha} \nu v \cdot A'}$
PAR	$\dfrac{A \xrightarrow{\alpha} A' \quad bv(\alpha) \cap fv(B) = bn(\alpha) \cap fn(B) = \varnothing}{A \mid B \xrightarrow{\alpha} A' \mid B}$
STRUCT	$\dfrac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad A' \equiv B'}{A \xrightarrow{\alpha} A'}$

Table 2.6: Labelled Reduction in the Applied Π-Calculus.

2. ∀ terms M, N: $(M =_E N)B$ iff $(M =_E N)B'$.

Extended processes A, A' are statically equivalent, if their frames are statically equivalent.

Labelled Bisimilarity. Labelled bisimilarity ($\approx_l$) is the largest symmetric relation $\mathcal{R}$ on closed extended processes, such that $A\,\mathcal{R}\,B$ implies:

1. $A \approx_s B$

2. if $A \rightarrow A'$ then $B \rightarrow^{\star} B'$ and $A'\mathcal{R}B'$ for some B'

3. if $A \xrightarrow{\alpha} A'$ and $fv(\alpha) \subseteq dom(A)$ and $bn(\alpha) \cup fn(B) = \varnothing$; then $B \rightarrow^{\star}\xrightarrow{\alpha}\rightarrow^{\star} B'$ and $A'\mathcal{R}B'$ for some B', where * denotes zero or more.

We write $A \Downarrow a$ when A can send a message on channel a, that is, when $A \rightarrow^{*}\equiv E[a\,\langle M\rangle\,.P]$ for some evaluation context $E[\;]$ that does not bind a.

An *observational bisimulation* is a symmetric relation $\mathcal{R}$ between closed extended processes with the same domain such that $A\ \mathcal{R}\ B$ implies:

1. if $A \Downarrow a$, then $B \Downarrow a$;

2. if $A \rightarrow^{*} A'$ and A' is closed, then $B \rightarrow^{*} B'$ and $A'\ \mathcal{R}\ B'$ for some B';

3. $E[A]\ \mathcal{R}\ E[B]$ for all closing evaluation contexts $E[\;\;]$.

Observational equivalence ($\approx$) is the largest such relation.

In [ABF17], the authors proved that observational equivalence is labelled bisimilarity, namely $\approx = \approx_l$.

2.2 Mathematical Background

This section establishes the necessary mathematical framework for the remainder of the work.

Finite Groups. A group $(\mathbb{G}, \cdot)$ is a pair where $\mathbb{G}$ is a non-empty set of elements, and $\cdot : \mathbb{G} \times \mathbb{G} \leftarrow \mathbb{G}$ is a binary relation verifying:

- *Associativity*: $\forall$ x, y and $z \in \mathbb{G}, (x \cdot y) \cdot z = x \cdot (y \cdot z)$
- *Identity element*: $\exists 1_{\mathbb{G}} \in \mathbb{G}$ such that $\forall x \in \mathbb{G}, x \cdot 1_{\mathbb{G}} = 1_{\mathbb{G}} \cdot x = x$.
- *Inverse element*: $\forall x \in \mathbb{G}, \exists$ a unique element $x^{-1} \in \mathbb{G}$ such that $x \cdot x^{-1} = x^{-1} \cdot x = 1_{\mathbb{G}}$.

Let $(\mathbb{G}, \cdot)$ be a group. $(\mathbb{G}, \cdot)$ is said to be finite if the elements of the set $\mathbb{G}$ are finite. Let $g \in \mathbb{G}$ and k is a natural number, we denote g^k to refer to the application of the relation $\cdot$, k times to the element g, i.e., $\underbrace{g \cdot \ldots \cdot g}_{k}$.

We say that g is a generator of a finite group $(\mathbb{G}, \cdot)$ if $\mathbb{G} = \{g^k \mid k \in \mathbb{Z}\}$ such that $\mathbb{Z}$ is the set of integers. When such an element exists, the group $(\mathbb{G}, \cdot)$ is called *cyclic*. When no confusion arises, we may refer to a group solely by its set, leaving the binary operation implicit in the notation. $\langle g \rangle = \mathbb{G}$ denotes a cyclic group $\mathbb{G}$ generated by g. The order of a cyclic group $\mathbb{G}$ corresponds to its number of elements.

Negligeable Function. Let $f : \mathbb{Z} \leftarrow \mathbb{R}$ be a function. f is said to be negligeable if $\forall n \geq 0, \exists N_n \in \mathbb{Z}$ such that $\forall x \geq N_n, |f(x)| \leq \frac{1}{x^n}$.

Discrete Logarithm Problem. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group. Given the pair $(g, g^x) \in \mathbb{G}^2$, the *Discrete Logarithm* (DL) problem requires to return the value x with non-negligeable probability. There exist groups in which the discrete logarithm problem appears computationally hard. Particularly, for certain algebraic structures, such as large prime-order subgroups of the multiplicative groups $(\mathbb{Z}_p, \times)$.

Finite Field. A finite field $(\mathbb{F}, +, \cdot)$ is a tuple where $(\mathbb{F}, +)$ is a finite commutative group with an identity element 0, $(\mathbb{F} \setminus \{0\}, \cdot)$ is a finite commutative group with an identity element 1, and the binary relation $\cdot$ is distributive with regard to the binary relation $+$. The discrete logarithm problem is hard, i.e., no polynomial algorithm is known for computing it, over a finite field with prime number of elements.

2.3 Cryptographic Primitives

The cryptographic literature offers a wide range of primitives, many of which serve as building blocks in our analyzed protocols. This section introduces the key primitives used in later chapters and outlines their properties.

2.3.1 Public key Encryption

A public key encryption scheme **PKE** = (**KeyGen**, **Enc**, **Dec**) is a tuple of algorithms. The key generation algorithm **KeyGen** takes as input a security parameter and returns a pair (**pk**, **sk**) of a public and secret key. The encryption algorithm **Enc** takes as input a message m and a public key **pk** and outputs a ciphertext C. If the encryption algorithm is additionally probabilistic, it takes a random coin r as an additional parameter. The decryption algorithm **Dec** takes as input a ciphertext C and a secret key **sk**, and outputs a message m or an error $\perp$. To each algorithm one or more input and output spaces are associated. We suppose that $(\mathcal{K}, \mathcal{M}, \mathcal{R}, \mathcal{C})$ are the associated keys, messages, coins and ciphertexts spaces respectively. The correctness of the encryption sckeme requires that for all keys (**pk**, **sk**) $\in \mathcal{K}$, all messages $m \in \mathcal{M}$ and all random coins $r \in \mathcal{R}$:

$$\mathbf{Dec}(\mathbf{Enc}(m, \mathbf{pk}; r), \mathbf{sk}) = m$$

Diffie-Hellman Key Exchange [DH76]. The security of the Diffie-Hellman key exchange relies on the computational hardness of solving the discrete logarithm problem over finite fields with prime number of elements. Let's consider g as a primitive element of such a finite field, i.e., a generator of the multiplicative group of the field. The Diffie-Hellman key exchange enables two parties Alice and Bob, with respective public keys g^a and g^b, to securely generate a shared secret g^{ab} over a public network, which can then be used for confidential communication. This shared secret is derived from their respective public keys, thus *binding* them cryptographically. One property of the exponentiation, which stems from field's definition, is its distributivity over multiplication. However, as discussed in the Introduction of this manuscript, security analysis of cryptographic protocols that incorporates this distributivity property becomes undecidable. We therefore cannot fully model all properties of the exponentiation and must instead exclude certain properties or work with approximations.

In ProVerif, the most commonly used equational theory for security protocols involving Diffie-Hellman keys is $\mathsf{exp}(\mathsf{exp}(g, a), b) = \mathsf{exp}(\mathsf{exp}(g, b), a)$ where g is a fixed generator. This equation describes the minimal relationship required for the protocol to work. That is, it ensures that Alice and Bob - each possessing the other's public key and their own private key - can compute the same shared secret. It is difficult to determine whether this equation alone is sufficient to prove the absence of attacks in a protocol model. For instance, in Chapter 3, we will see that it is insufficient - relying solely on this equation in the model leads us to miss several attacks where we need more than two exponentiations. However, in Chapter 4, we will see that all the results obtained by modeling the exponentiation as an associative and commutative symbolic operation in Tamarin lead to the same conclusions about the protocol's security when using simply the ProVerif's equation. Several existing works in the literature focus on precise modeling of modular exponentiation. For instance, multiple studies address the incorporation, into symbolic models, of small order subgroups. Notable examples include the works of [CJ19a] in Tamarin and [BDP15] in ProVerif.

RSA Cryptosystem [RSA78]. RSA is a foundational public-key cryptosystem, proposed by Ronald Rivest, Adi Shamir, and Leonard Adleman in 1978. Its encryption scheme derives security from the hardness of integer factorization, making it a cornerstone of modern cryptography. The key generation algorithm, the encyption and decryption algorithms are described in Algorithm 1. RSA encryption is an homomorphic encryption scheme. Specifically, the encryption of the product of two messages $\mathbf{Enc}(m \cdot m', \mathbf{pk})$ equals the product of their encryptions $\mathbf{Enc}(m, \mathbf{pk}) \cdot \mathbf{Enc}(m', \mathbf{pk})$. To our knowledge, no existing work specifically targets the precise symbolic modeling of RSA encryption - it is typically modeled like any other deterministic public-key encryption scheme, i.e., $\mathsf{adec}(\mathsf{aenc}(m, \mathsf{pk}(sk)), sk) = m$ where aenc, adec, and pk are function symbols.

Algorithm 1 RSA cryptosystem where φ denotes Euler's totient function and gcd denotes the greatest common divisor.

1. RSA KeyGen

```
select two different large primes $p$ and $q$
$N \leftarrow p \cdot q$
choose integer $e$ such that $1 < e < \varphi(N)$ and $\gcd(e, \varphi(N)) = 1$
$d \leftarrow e^{-1} \pmod{\varphi(n)}$
$\mathsf{sk} \leftarrow (p, q, d)$
$\mathsf{pk} \leftarrow (N, e)$
return $(\mathsf{pk}, \mathsf{sk})$
```

2. RSA Enc(m, pk)

```
$(N, e) \leftarrow \mathsf{pk}$
$c \leftarrow m^e \pmod{N}$
return $c$
```

2. RSA Dec(c, sk)

```
$(\varphi_N, d) \leftarrow \mathsf{sk}$
$m \leftarrow c^d \pmod{N}$
return $m$
```

ElGamal Cryptosystem [ElG84]. The ElGamal public key encryption scheme was introduced by Taher Elgamal in 1984. Its security is based on the hardness of the discrete logarithm problem in multiplicative cyclic groups with large prime order. The key generation algorithm, the encyption and decryption algorithms are described in Algorithm 2. ElGamal encryption is also homomorphic, a property that enables its use in numerous electronic voting protocols. However, similar to RSA, we are not aware of any existing work that provides a less abstract symbolic modeling of this encryption scheme. In the next chapter, we propose a more precise model of ElGamal cryptosystem that leverages the specifics of the keys modular exponentiation.

Algorithm 2 ElGamal cryptosystem.

```
1. ElGamal KeyGen
    $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p$
    $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
    return $(\mathsf{pk}, \mathsf{sk})$
2. ElGamal Enc(m, pk)
    $\mathsf{r} \xleftarrow{\$} \mathbb{Z}_p$
    return $(g^r, m\mathsf{pk}^r)$
3. ElGamal Dec($(c_1, c_2)$, sk)
    $m \leftarrow c_2 \cdot {c_1}^{-\mathsf{sk}}$
    return $m$
```

2.3.2 Signatures Schemes.

Digital signatures serve as the electronic counterpart to handwritten signatures, providing cryptographic proof that a message originates from a specific individual or entity. They function similarly to traditional signatures while offering equivalent security properties. Formally, a digital signature scheme consists of three algorithms (**SGen**, **Sign**, **Verify**). The key generation algorithm **SGen** takes as input a security parameter and returns a pair (**pk**, **sk**) of a public and secret key. The signature algorithm **Sign** takes as input a message m and a secret key **sk** and outputs a signature σ. The verification algorithm **Verify** takes as input a signature σ, a public key **pk**, and a message m, and outputs either 0 or 1. To each algorithm one or more input and output spaces are associated. We suppose that $(\mathcal{K}, \mathcal{M})$ are the associated keys and messages spaces respectively. The correctness of the signature scheme requires that for all keys (**pk**, **sk**) $\in \mathcal{K}$, all messages $m \in \mathcal{M}$:

$$\mathbf{Verify}(\mathbf{Sign}(m, \mathbf{sk}), \mathbf{pk}, m) = 1$$

The signature scheme should ensure *existantial unforgeablity*, that is, only the person holding the secret key can generate a valid signature for the associated public key. The standard way to model signatures in the symbolic model is using the equation $\mathsf{verify}(\mathsf{sign}(m, sk), m, \mathsf{pk}(sk)) = \mathtt{true}$. Jackson et al. [JCCS19] point out that this equation is stronger than existential unforgeability and using it in models may cause to overlook certain attacks. They therefore developed more precise signature models in TAMARIN to account for subtle signature scheme behaviors.

2.3.3 Authenticated Encryption with Associated Data.

An AEAD is a symmetric authenticated encryption algorithm. The encryption algorithm $\mathsf{AEAD.Enc}(k, N, H, M)$ takes as input a key k, a nonce N, a header H, and a message M, and outputs a ciphertext C. The decryption algorithm $\mathsf{AEAD.Dec}(k, N, H, C)$ takes as input a key k, a nonce N, a header H and a ciphertext C, and outputs a message M or $\perp$. We denote by SE an **unauthenticated symmetric encryption scheme**. $\mathsf{SE.Enc}(k, N, M)$ takes as input a key k, a nonce N, and a plaintext M, and outputs a ciphertext C, while $\mathsf{SE.Dec}(k, N, C)$ takes as input a key k, a nonce N, and a ciphertext C, and outputs a plaintext M.

The commonly used equation for modeling symmetric encryption in the symbolic model is $\mathsf{sdec}(\mathsf{senc}(m, sk), sk) = m$. This equation remains highly abstract and does not capture the specific details of the underlying symmetric scheme. In [CDJZ23b], the authors introduced the first automated analysis technique for protocols using AEAD schemes. Their models systematically identifies attacks that leverage subtleties of the particular AEAD construction used.

2.3.4 Key Encapsulation Mechanisms

A Key Encapsulation Mechanism KEM consists of three algorithms: $\mathsf{KEM.gen}()$ which generates the key pair (s, S), $\mathsf{KEM.encaps}(\mathsf{S})$ a probabilistic encapsulation algorithm, which takes as input a KEM public key S, and outputs a secret key k and a ciphertext ct. We also use the definition of KEM.encaps introduced in [HNS$^+$21] to make the algorithm deterministic, by adding an interface to provide random coins R as additional input to the algorithm, i.e., $\mathsf{KEM.encaps}(\mathsf{S}, R)$. Finally, $\mathsf{KEM.decaps}(\mathsf{s}, ct)$ is the deterministic decapsulation algorithm, which takes as input a KEM private key s, and a ciphertext ct, and outputs a secret key k. Details about the symbolic snalysis of KEM-based protocols are given in Section 5.1.2 of Chapter 5.

2.4 Conclusion

In this section, we introduced applied Π-Calculus, the formal language used to model all our analyzed protocols. We also introduced the cryptographic primitives that serve as building blocks for the analyzed protocols. For each primitive, we presented the most commonly used equations for their symbolic modeling. While standard equations remain highly abstract (relying on perfect cryptography assumptions), existing research efforts aim to incorporate cryptographic subtleties into symbolic models - both to prevent missed attacks and reduce abstraction. The following chapter aligns with this research direction.

Remark:

Throughout this manuscript, we use the terms adversary, attacker, and dishonest participant interchangeably to refer to any protocol participant that deviates from the protocol's specifications. Conversely, honest or trustworthy denotes a participant that adheres strictly to the protocol specification. We say a protocol is secure if it satisfies all its defined security properties; conversely, terms like insecure, broken, or attack indicate that at least one security property is violated. A security property is verified or satisfied if it holds under formal analysis, and not verified or falsified if it fails to hold. Finally, adversary model and threat model are used interchangeably.

Chapter 3
Automated Discovery of Subtle Attacks on Protocols using Mix-Networks

> The beauty of mathematics only shows itself to more patient followers.
>
> *Maryam Mirzakhani*

Contents

Chapter Summary

In this chapter, we propose a more detailed formal model of exponentiation and re-encryption Mix-Networks (Mix-Nets) in the applied Π-Calculus, the language used by ProVerif. We show that using this model, we can automatically discover attacks based on the incorrect use of the Mix-Net, thus, helping protocol designers avoiding these and similar attacks in the future. In particular, we find attacks on four cryptographic protocols using ProVerif: we show that an electronic exam protocol, two electronic voting protocols, and the "Crypto Santa" protocol do not satisfy the desired privacy properties. We then fix the vulnerable protocols by adding missing Zero knowledge proofs (ZKPs) and re-analyze the resulting protocols using ProVerif. Again, in addition to the common abstract modeling of the ZKP, we also propose a new model corresponding to weak (malleable) ZKPs. We show that in this case all attacks persist, and that we find again these attacks automatically.

3.1 Introduction

Mix-Net is a mechanism, based on public key cryptography, proposed by David Chaum in 1981, that aims to "*hide the correspondence between the items in its inputs and those in its outputs*" [Cha81]. The idea is rather simple: suppose we have two senders and one receiver, and we want the receiver to not trace a received message back to its sender. If each sender directly sends an encrypted message with the public key of the receiver to the receiver themselves, it is obvious that the receiver would know which message corresponds to which sender. Furthermore, if the receiver publishes the decrypted messages or the messages' space is small, and the employed encryption algorithm is deterministic, anyone can easily associate a plaintext with a sender. Instead of doing so, and assuming that the encryption procedure is done using a randomly generated seed, both senders re-encrypt their encrypted message with the public key of another entity, the Mix-Net. The Mix-Net decrypts the incoming ciphertexts and outputs the resulting plaintexts in a random order. Upon receiving the messages from the Mix-Net, the receiver is unable to draw any conclusions about the relation between a message and a sender. In [Cha81], Chaum showed how this technique can be used to make electronic mails, return addresses and digital pseudonyms *untraceable*, and how we can use a series of Mix-Nets to avoid relying on a single Mix-Net server.

These types of Mix-Nets will bear the name of *Decryption Mix-Net* since the mixing servers decrypt the items of their inputs.

Since their introduction, many Mix-Net constructions have been proposed, analyzed and integrated into various cryptographic protocols [AW07, AKTZ17, KMW12, ANJ+24, Wik04b, WG06, BHM20, GJJS04]. In [PIK93], Park *et al.* presented the *Re-encryption Mix-Net*. These Mix-Nets rely on malleability properties of the encryption scheme, giving the possibility to re-encrypt a list of ciphertexts resulting in a new list of ciphertexts, without knowing or changing the associated plaintexts. In [HS11], Haenni *et al.* presented a different form of a Mix-Net, the so-called *Exponentiation Mix-Net*. These Mix-Nets create, from a list of ElGamal public keys [ElG85], a new shuffled list of anonymized public keys, which can no longer be associated to individual parties, but can still be used to verify signed messages or encrypt data.

Considering their importance in providing anonymity to the communicating parties, Mix-Nets play a major role in building systems where privacy is a key security requirement, such as e-voting systems [Adi08, CGG22, CRST15a, HKLD17, KMST16, HKL23, HS11] or e-exams [GLR14, RGL22]. In the context of voting, they are used to hide the link between voters and their votes to guarantee the voters' privacy, as typically the protocol publishes the list of the decrypted votes at the end to achieve verifiability. In such a protocol, any possibility to link a voter to his decrypted plaintext vote immediately breaks privacy. Voting protocols using Mix-Nets have been even employed in real political elections, for example in Estonia [IVX23], in the Australian state of Victoria [CRST15b] or in Norway [Gjø12]. In the context of electronic exams, Mix-Nets are used to hide the relation between the candidates and their exam tests or exam answers to guarantee candidates' privacy and the impartiality of the marking procedure, e.g., the C-Rex [RGL22] and Remark! [GLR14] exam protocols.

In [DGK+14], the Remark! exam protocol, which is based on the exponentiation Mix-Net, was analyzed and proven secure using ProVerif. As explained in the previous chapter, in the symbolic model the properties of cryptographic primitives are usually modeled using equations, e.g., the equation $\mathsf{dec}(\mathsf{enc}(m, k), k) = m$ can be used to model a simple deterministic symmetric encryption scheme, where the functions dec and enc represent decryption and encryption operations, respectively. These equations can fail to take into account subtle behaviors of primitives which can lead to miss certain classes of attacks. For example, the model of Remark! in [DGK+14] abstracts away completely the implementation of the Mix-Nets. Hence, they missed an attack on the Mix-Net described in [RGL22] that exploits the details of the exponentiation process. This attack was found manually and works as follows: an attacker submits a modified version of the public key of a targeted user as their own key to the Mix-Net. This enables the attacker to link both keys after the mixing, breaking the anonymity of the targeted user.

Similar attacks on Mix-Nets had already been described in 1994 [Pfi94], long before exponentiation mixnets [HS11] where proposed in 2011. Still, it took a decade for [RGL22] to (apparently independently) discover that such use of Mix-Nets was insecure.

3.1.1 Our Contributions

We show that it is possible to model Mix-Nets more precisely in the symbolic model. This allows us to automatically find previously missed (but known) attacks, to discover new attacks, or to verify properties of protocols against a more powerful attacker that can exploit weaknesses of the Mix-Net. Our contributions are the following:

- We propose a detailed model of Haenni's [HS11] exponentiation Mix-Net, taking into account details of the exponentiations.
- Our refined model includes a new, more precise model of ElGamal encryption and signatures, which includes the exponentiation operations and is of independent interest as it can be used to model more precisely protocols that use ElGamal encryption or signatures, but no exponentiation Mix-Nets. To the best of our knowledge, until now symbolic models of ElGamal encryption completely abstracted away the exponentiations, hence potentially missing attacks.
- We then show that this new model of the exponentiation Mix-Net together with the model of ElGamal can be used to automatically analyze protocols using this Mix-Net. For this, we model and analyze three protocols in ProVerif: an e-voting protocol [HS11], an e-exam protocol [GLR14], and the Crypto Santa protocol [Rya15].
    - We give a formal model and analysis of the electronic voting protocol from [HS11]. Our analysis shows that it is vulnerable to a privacy attack (similar to the one from [RGL22]) due to a weakness in the Mix-Net, and the lack of Zero-Knowledge Proofs on the voters' side.
    - We are able to automatically find the attack from [RGL22] on the exam protocol. Interestingly, Amin et al. [RGL22], despite identifying this attack manually, used a simple formal model in ProVerif which was incapable of detecting the vulnerability.
    - We provide the first formal model and analysis of the Crypto Santa protocol, which turns out to be not vulnerable due to use of appropriate ZKPs by the participants.

- To fix the identified attacks we add the lacking ZKPs to the voting and exam protocols. We check the protocols again, using the standard symbolic modeling of ZKPs, and show that the corrected protocols are secure.

- The standard symbolic modeling corresponds to ideal ZKPs. We propose a model for weaker Zero-Knowledge Proofs (which are common [DMWG23], although they are known to be vulnerable [BPW12]), and can show that they are insufficient in this case, as all attacks persist. To the best of our knowledge, these concrete attacks have not been described in the literature. Even the Crypto Santa protocol becomes vulnerable when using these weak proofs, yet the protocol does not specify which ZKPs should be used. Again, the model of weak ZKPs is of independent interest, as it can be used for any protocol using ZKPs of ElGamal keys, even if there is no Mix-Net.

- We also propose a refined model for re-encryption Mix-Nets, and analyze the Estonian e-Vote protocol [IVX23]. As the protocol does not use any ZKP by the voters, it is vulnerable to attacks on the Mix-Net, which we can automatically detect (similar attacks were found manually before, e.g., [Mül22] and which were judged to be "*currently outside the scope of automated tools*" in [BBMP24]). We verify the protocol with added weak or strong ZKPs, and are able to show that only the strong ZKPs are sufficient to avoid attacks.

Remark:

In this chapter, we only consider ZKPs on the participants' side to avoid attacks by dishonest participants. As the Mix-Nets themselves can be dishonest, they can be carried out with other ZKPs for correct "mixing" (or shuffle) to guarantee that no inputs are added, omitted or altered which is used to provide publicly verifiable transcripts that prove that the outputs are indeed a shuffled list of the inputs. We omit this part as we only consider honest Mix-Nets in our work. For more details about the latter proofs, see [TW10, BG12].

3.1.2 Related Work

The first formal model of Mix-Nets was provided in 2003 by Wolff *et al.* [WSF$^+$03]. They formalized the Mix-Net and its components using the CSP process algebra and the FDR model-checker. The formal analysis was conducted considering a passive attacker, and the Mix-Net model was not specific to a particular type of Mix-Net, but rather general. At this time, this tool was only able to deal with simple dedicated equational theory describing public key cryptography. In

2014, Stathakidis *et al.* [SSH14] proposed a formal model for *Ximix*, a Java implementation of re-encryption MIX-NET, suitable for automation based on CSP process algebra and the FDR model checker. They model and analyze the protocol in the presence of an intruder based on Roscoe and Goldsmith's perfect Spy [Ros05]. The main security property analyzed was the robustness of Ximix, which is a functional property. Moreover, they do not model any mathematical properties of the MIX-NET. Also in 2014, Küsters *et al.* [KTV14] provided the first computational formal security analysis of decryption MIX-NET with random partial checking. In their paper, they focus on accountability, where misbehavior should be detectable and the responsible of the misbehavior should be blamed. Their analysis in the computational model is not automated. There are other works on the verification of MIX-NETS, e.g., in [HGS21], where the authors provide a mostly manual proof of the Verificatum MIX-NET, but to the best of our knowledge there is none targeting directly exponentiation MIX-NETS.

Nonetheless, the most prevalent and standard method of modeling MIX-NETS for symbolic verification of security protocols, is through anonymous channels, as demonstrated by the overwhelming majority of works from the literature [BS16, Swi22, BCG$^+$18, Bel20, CGT17]. This is not surprising, given that MIX-NETS are designed to establish a completely anonymous channel between incoming and ongoing messages. In a nutshell, the MIX-NET is modeled abstractly by sending the incoming messages over a private channel concurrently, thus without fixing the order, and reading them back from that same channel.

Our work in the symbolic model using PROVERIF allows us to automatically discover flaws or verify protocols using MIX-NETS. Compared to previous works in the symbolic model, our models evolve from a complete abstraction based on the functionality of MIX-NETS, to a more precise and less abstract models of exponentiation and re-encryption MIX-NETS.

Equivalence Properties and Automated Tools. There exist several automated protocol verification tools handling equivalence properties, but we chose PROVERIF as DEEPSEC [CKR18] cannot handle our equational theory, and, although TAMARIN [BCDS17] supports Diffie-Hellman exponentiation, the built-in exponentiation operator cannot appear inside a user defined equation. In the case of exponentiation MIX-NETS we need to define encryption and signature schemes using the Diffie-Hellman keys, hence we need to re-use the exponentiation operator in the equations modeling encryption and signature. Previous work mostly focused on improving the modeling of exponentiation in isolation [CJ19b], but we need to connect both exponentiation and encryption. Computational tools such as EasyCrypt [BGZBH11] or CryptoVerif [Bla09] should be able to detect such attacks, they however provide a significantly lower level of automation.

Refined Modeling of Cryptographic Primitives. This work follows a line of works which try to refine the modeling of cryptographic primitives in symbolic models and thereby to reduce the gap to computational models, while keeping a high level of automation. Similar work has been done for other primitives, for example for digital signatures [JCCS19], Diffie-Hellman exponentiation in weak groups [CJ19b], hash functions [CCD$^+$23b], authenticated encryption with additional data (AEAD) [CDJZ23a], and key encapsulation mechanisms (KEM) [CDM24]. In all these papers, the authors proposed more detailed and more realistic models of cryptographic primitives in the symbolic framework, to be able to identify protocol flaws exploiting weaknesses of the cryptographic primitives. In [JCCS19], the authors refine the existing Tamarin models of signature schemes so they discover attacks exploiting for example the fact that in some schemes a signature potentially verifies against multiple different keys. In [CJ19b], the authors propose a novel extension of the symbolic model of Diffie-Hellman groups for Tamarin enabling them to capture a large family of attacks exploiting weak groups that were previously outside the symbolic model. In [CCD$^+$23b], they propose a methodology to systematically discover attacks using Tamarin and ProVerif that exploit weaknesses in widely deployed hash functions, such as length extension attacks. They automatically find known attacks, but also new variants of these attack on several protocols. In [CDJZ23a], the authors provide an automated analysis method in Tamarin for protocols that use AEAD. This method allows them to systematically find attacks that exploit the subtleties of the specific type of AEAD used, as different schemes have different weaknesses.

Link to Computational Models. In symbolic models, all negligible probabilities are abstracted away. For example, if there is only a negligible probability to generate twice the same random value, or for the adversary to break an encryption scheme, in a symbolic model this will simply be impossible. It has been shown that under certain assumptions this can nevertheless give computational guarantees (an approach called “computational soundness”, see e.g. [BBU13, MW04]). There are also works on symbolic models that allow non-negligible probabilities in the control flow, i.e., where agents can chose among different actions with given probabilities (e.g., [CCK22]), however there is currently no tool support for this. As the goal of our work is to automatically identify attacks, we use standard symbolic models, where automated tools exist.

Outline. We start by describing decryption, re-encryption and exponentiation Mix-Nets and detail known attacks from the litterature against those types of Mix-Nets. In Section 3.3, we present our refined models, in the symbolic approach, of ElGamal cryptosystem, DSA signatures and zero knowledge proofs.

In Section 3.4, we present our new formal models for exponentiation and re-encryption MIX-NETS. We present, as well, the result of our analysis with PROVERIF when verifying anonymity. In Section 3.5, we apply those models to four different protocols using MIX-NETS to guarantee the privacy of the participants. We conclude and discuss future work in Section 5.5.

3.2 Overview of Known Attacks on MIX-NETS and Countermesures

In this section, we describe known attacks from the literature against the exponentiation and re-encryption MIX-NETS, and their countermeasures. Even though we are not particularly targeting decryption MIX-NETS, we will nonetheless begin by providing an overview of an attack against this type of MIX-NET when instantiated with the RSA cryptosystem, since it is the first chronologically exhibited attack against the MIX-NETS and since it shares some similarities with the attacks against both exponentiation and re-encryption MIX-NETS.

3.2.1 Decryption MIX-NETS

The following is based on the description of the untraceable electronic mail system given by David Chaum in [Cha81]. For sake of simplicity, we suppose that there is only one mixing server $\mathsf{M_D}$ with $(\mathsf{sk}_{\mathsf{M_D}}, \mathsf{pk}_{\mathsf{M_D}})$ as secret and public key pair, a receiver R with $(\mathsf{sk_R}, \mathsf{pk_R})$ as secret and public key pair, and three senders S_1, S_2 and S_3. We also suppose that $\mathsf{enc}(m, r, \mathsf{pk}_i)$ and $\mathsf{dec}(c, \mathsf{sk}_i)$ denote respectively the encryption of a message m, sealed with the randomness r, with the public key of an entity i and the decryption of a ciphertext c with the secret key of the entity i. The protocol works as follows. Each sender S_j encrypts their message m_j sealed with r_j, with the public key of the receiver R and obtains $c_j = \mathsf{enc}(m_j, r_j, \mathsf{pk_R})$. Then, S_j re-encrypts the resulted ciphertext c_j sealed with the randomness r'_j, with the public key of the MIX-NET $\mathsf{M_D}$, obtains $c'_j = \mathsf{enc}(c_j, r'_j, \mathsf{pk}_{\mathsf{M_D}})$ and sends c'_j to $\mathsf{M_D}$. $\mathsf{M_D}$ receives the three ciphertexts, decrypts each one with their secret key, re-arranges the plaintexts in lexicographic order and sends them to R. Chaum emphasized that MIX-NETS must check inputs for uniqueness and that no input should be repeated more than once because otherwise the correspondence between the inputs and outputs is revealed for that repeated item (as a redundant input yields in a redundant output). Decryption MIX-NET was defined with regard to any public key cryptosystem such as RSA.

Attack on the RSA Implementation of Decryption MIX-NET [PP90].

Let us consider the RSA instantiation of the decryption MIX-NET by Chaum. Consider the RSA setup as defined in Section 2.3.1, but instead of encrypting

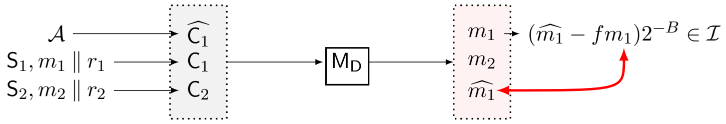


Figure 3.1: Attack against the RSA implementation of decryption Mix-Nets, where $\widehat{\mathsf{C}_1} \equiv f^{\mathsf{e}}\mathsf{C}_1 \pmod{\mathsf{N}}$ and $\mathcal{I} = \{-2^b + 1, \dots, f(2^b - 1)\}$.

simply a plaintext m, a sender S_i encrypts their message m_i concatenated with a randomly generated coin r_i both with fixed bits-size. Let B and b be the bit-length of m_i and r_i respectively. We suppose that upon receiving the ciphertexts from the sender, M_D decrypts them with their secret key and retreives m_i from $r_i \parallel m_i$. The attack described in [PP90] makes use of the multiplicative homomorphic property of RSA and it is depicted in Figure 3.1. The attacker has access to the list of the Mix-Nets' input $\langle \mathsf{C}_1, \mathsf{C}_2 \rangle$ of the two senders. The attacker $\mathcal{A}$, which corresponds to sender S_3, wants to learn the plaintext corresponding to a specific ciphertext C_1 sent by S_1. $\mathsf{C}_1 \equiv (r_1 \parallel m_1)^{\mathsf{e}} \pmod{\mathsf{N}}$. $\mathcal{A}$ can generate a small bit-sized number f. $\mathcal{A}$ encrypts f with (e, N) the public key of M_D, multiplies the resulting ciphertext with C_1 and sends the ciphertext $\widehat{\mathsf{C}_1} \equiv (f(r_1 \parallel m_1))^{\mathsf{e}} \pmod{\mathsf{N}}$ to the Mix-Net as their own ciphertext. When the output list of the encrypted messages is published, $\mathcal{A}$ knows that there is two output values m_i and $\widehat{m_i}$ such that $(\widehat{m_i} - fm_i)2^{-B}$ belongs to $\mathcal{I} = \{-2^b + 1, \dots, f(2^b - 1)\}$. Hence, $\mathcal{A}$ looks for pairs verifying this condition in the output list. This way $\mathcal{A}$ identifies the plaintext m_1 of the sender S_1.

Technical Detail

We have $\mathsf{C}_1 \equiv (r_1 \parallel m_1)^{\mathsf{e}} \pmod{\mathsf{N}} \equiv (r_1 \cdot 2^B + m_1)^{\mathsf{e}} \pmod{\mathsf{N}}$ and $\widehat{\mathsf{C}_1} \equiv (\widehat{r_1} \cdot 2^B + \widehat{m_1})^{\mathsf{e}} \pmod{\mathsf{N}} \equiv (fr_1 \cdot 2^B + fm_1)^{\mathsf{e}} \pmod{\mathsf{N}}$. This implies that $\widehat{m_1} - fm_1 \equiv 2^B(fr_1 - \widehat{r_1}) \pmod{\mathsf{N}}$. Hence $(\widehat{m_1} - fm_1)2^{-B} \equiv fr_1 - \widehat{r_1} \pmod{\mathsf{N}}$. And since b is the bit-length of r_1 and $\widehat{r_1}$, we have that $fr_1 - \widehat{r_1} \in \{-2^b + 1, \dots, f(2^b - 1)\} = \mathcal{I}$.

Remark:

This attack can be seen as a form of a choosen ciphertext attack against RSA with the Mix-Net acting as a decryption oracle. These types of attacks against RSA were already known to the cryptography community since 1982 with the work of G. I. Davida [Dav82]. The subtle difference with the attack described in [PP90] is that the random string in a decrypted message is not known to the attacker (the Mix-Net only outputs the message). In a similar vein, exploiting the format and the bit-length of the encrypted message as a padding string concatenated with a block of data, Daniel Bleichenbacher described, later in 1998, successful chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1 in [Ble98].

3.2.2 Re-Encryption Mix-Nets

The main disadvantage of decryption Mix-Nets is the robustness of the network. First, the length of the ciphertexts that each sender sends is very large. It grows proportionally to the number of Mix-Nets servers. Secondly, the incoming messages have to be decrypted and passed on in the correct order. If only one Mix-Net server is inactive in the phase of mixing, the message cannot be transmitted. To overcome those limitations, Park *et al.* proposed the re-encryption Mix-Net in [PIK93]. In re-encryption Mix-Nets, both the input and output lists are ciphertexts encrypted within the same public-key encryption scheme. These Mix-Nets put to positive use the homomorphic properties of the used cryptosystem which allows the servers to re-randomize ciphertexts, thereby re-encrypting the corresponding plaintexts. Thus, there is no a pre-defined order among the mixing servers and the length of the ciphertexts is irrelevent to the number of the servers. Let us consider the ElGamal setup as defined in Section 3.3.1. We have a list of n ciphertexts $\langle \mathsf{C}_i \rangle = \langle \mathsf{C}_1, \dots, \mathsf{C}_n \rangle$ such that $\mathsf{C}_i = (\mathsf{g}^{r'_i}, m_i \mathsf{h}^{r'_i})$, where m_i a plaintext, and m mix servers M_j. The first mix server M_0 takes the original list of ciphertexts $\langle \mathsf{C}_i \rangle$, and for each ciphertext C_i generates a fresh random coin $r_i \in \{0, \dots, q-1\}$ and re-randomizes the ciphertext with r_i to obtain $\mathsf{C}'_i = (\mathsf{g}^{r'_i}\mathsf{g}^{r_i}, m_i \mathsf{h}^{r'_i}\mathsf{h}^{r_i})$. Then M_0 permutes the resulting terms using a secret random order and sends the new list to the next mix server. The following mix servers repeat these same steps. The last mix server M_{m-1} outputs a list of ciphertexts which encrypt the permuted input messages, initially chosen by the senders, encrypted using different and secret random values. Due to their construction, re-encryption Mix-Net outperforms the decryption Mix-Net in terms of efficiency [RA12].

Attack on the Re-Encryption Mix-Net [Pfi94].

In 1994, Birgit Pfitzmann [Pfi94] described the attack depicted in Figure 3.2 against re-encryption Mix-Nets in the context of electronic voting. In this case encrypted ballots are sent to a Mix-Net. Once the ballots are mixed, the authority decrypts all the mixed encrypted ballots, and publishes the plaintext values in order to ensure verifiability properties. The attack considers a malicious voter $\mathcal{A}$ participating in the protocol. The attacker has access to the list of the Mix-Nets' input $\langle \mathsf{C}_i \rangle$ of all participants. The attacker wants to learn the plaintext corresponding to a specific ciphertext $\mathsf{C}_k \in \langle \mathsf{C}_i \rangle$ such that $\mathsf{C}_k = (\mathsf{g}^{r_k}, m_k \mathsf{h}^{r_k})$. $\mathcal{A}$ can generate $a \in \{1, \dots, q-1\}$, raise both components of C_k to the power a, and send the ciphertext $\widehat{\mathsf{C}_k} = (\mathsf{g}^{r_k a}, (m_k \mathsf{h}^{r_k})^a)$ to the Mix-Net as its vote. When the

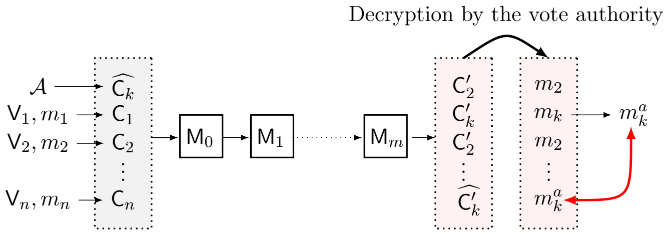


Figure 3.2: Attack against re-encryption Mix-Nets, where $\widehat{\mathsf{C}_k} = (\mathsf{g}^{r_k a}, (m_k \mathsf{h}^{r_k})^a)$.

output list of encrypted messages is sent to the authority, all plaintexts m_i are published, and $\mathcal{A}$ knows that there is one output value m_i that is equal to m_j^a. Hence, $\mathcal{A}$ looks for this collision by raising each plaintexts to the power of a and looking for a match with the others. This way $\mathcal{A}$ can identify the plaintext m_i of the victim.

3.2.3 Exponentiation Mix-Nets

In [HS11], Haenni *et al.* have presented *Exponentiation Mix-Nets*, which create, from a list of ElGamal public keys, a new shuffled list of anonymized public keys, which can no longer be associated to individual parties.

Let us consider the same ElGamal setup described in Section 3.3.1. We have a list of n public keys $\langle \mathsf{pk}_i \rangle = \langle \mathsf{pk}_1, \ldots, \mathsf{pk}_n \rangle$ such that $\mathsf{pk}_i = \mathsf{g}^{sk_i}$ and m mix servers M_j. The first mix server M_1 takes the original list of public keys $\langle \mathsf{pk}_i \rangle$, generates a fresh random $r_1 \in \{0, \ldots, q-1\}$ and computes $\langle \mathsf{pk}_i^{r_1} \rangle$. Then M_1 permutes the resulting terms with a secret shuffled order $\langle \mathsf{pk}_{\pi_1(i)}^{r_1} \rangle$ where π_1 is the permutation of indexes applied by M_1. Then M_1 sends $\langle \mathsf{pk}_{\pi_1(i)}^{r_1} \rangle$ to the next mix server along with g^{r_1}. The following mix servers repeat these same steps as required. The last mix server M_m outputs the list: $\langle \mathsf{g}^{sk_{\pi(1)} \cdot r}, \mathsf{g}^{sk_{\pi(2)} \cdot r}, \ldots, \mathsf{g}^{sk_{\pi(n)} \cdot r} \rangle$ along with g^r where: $\pi = \prod_{i=1}^{m} \pi_i$ and $r = \prod_{i=1}^{m} r_i$. Each party in possession of their secret key sk_i should be the only one able to identify her pseudonym from the final list published by the last mix server by computing $(\mathsf{g}^r)^{sk_i} = \mathsf{pk}_i^r$.

Attack on the Exponentiation Mix-Net [RGL22].

Even though exponentiation Mix-Nets were designed nearly two decades after the discovery of above attack against the re-encryption and decryption Mix-Nets, there is an attack against them. Assume an attacker $\mathcal{A}$ that has access to the list of public keys $\langle \mathsf{pk}_i \rangle$ of all participants. They want to learn the pseudonym generated by the mix servers for a specific public key pk_k. $\mathcal{A}$ can generate $a \in \{1, \ldots, q-1\}$, raise pk_k to the power a and append term $(\mathsf{pk}_k)^a$ to the list of public keys as their own public key to be mixed. When the final list of pseudonyms is

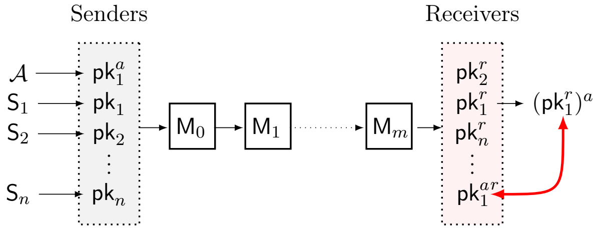


Figure 3.3: Attack against Exponentiation MIX-NETS.

published, they raise each pseudonym to the power of a and look for a match. The attack, as illustrated in Figure 3.3 for a list of n public keys, exploits the commutative property of exponentiation: $((\mathsf{g}^{sk_i})^a)^r = ((\mathsf{g}^{sk_i})^r)^a$. This attack (a more specific variant of it) was described by Rakeei *et al.* [RGL22] in 2022, but it can be seen as a variant of the attack firstly described by Pfitzmann [Pfi94] in 1994.

3.2.4 Countermesures

To overcome the attacks described above against exponentiation and re-encryption MIX-NETS, the senders are required to provide *Zero-Knowledge Proofs* (ZKPS) of knwoledge [SP07, RGL22, Jak98]. A ZKP allows a party to show to another party that a mathematical statement is true without revealing anything other than the truth of the statement itself. A specific class of ZKPS are proofs of knowledge, in which the prover demonstrates the knowledge of the preimage $x \in X$ of a public value $y = \phi(x) \in Y$, where ϕ is supposed to be a one way function. These proofs are in particular used to prove knowledge of the Discrete Logarithm (DL) x of $y = g^x$ belonging to a multiplicative finite group G_q with a generator g of order q. Moreover, interactive proofs between provers and verifiers can be turned into non-interactive ones using the Fiat-Shamir heuristic [FS87]. Such ZKPS are, for example, used to avoid attacks against MIX-NETS by requiring the senders to prove that they know the secret key associated to their claimed public key in case of the exponentiation MIX-NET and the knwoledge of randomness used when encrypted a message using ELGAMAL in the case of re-encryption MIX-NETS.

Intuitively, adding a ZKP of the possession of the secret information along with the messages fixes the attacks mentioned in Section 3.2. In practice, we need to add Non-Interactive ZKPS in order to avoid burdening the protocol with other exchanged messages. The Fiat-Shamir transformation appears to be the most efficient construction of non-interactive ZKPS [BPW12]. However, two variants of the Fiat-Shamir transformation appear in the literature. In [BPW12], Bernhard *et al.* distinguish a weaker and a stronger variant. Both variants begin with the prover making a commitment. The stronger variant hashes both the commitment and the statement to be proved, while the weak variant hashes only the commitment. The weak version is subject to an attack which allows a malicious party to “fake” a proof as follows [BPW12]. To create a proof, an honest prover, having as public key $pk = g^{sk}$, picks a random $a \in \{1, \ldots, q-1\}$, computes $A = g^a$ and then, hashes A to create a challenge $c = \mathcal{H}(A)$. Finally, the prover computes $f = a + c \cdot sk$. The proof corresponds to the pair (c, f) and the verification procedure consists in checking whether c is equal to $\mathcal{H}(g^f \cdot (pk)^{-c})$ or not. For an honestly generated proof the verification procedure succeeds since we have $\mathcal{H}(g^f \cdot pk^{-c}) = \mathcal{H}(g^{a+c\cdot sk} \cdot g^{-sk\cdot c}) = \mathcal{H}(g^a) = c$. A malicious prover picks A' a random element from G_q, f' a random exponent, computes $c' = \mathcal{H}(A')$ and the verification of the proof (c', f') succeeds for $pk' = (g^{f'} \cdot A'^{-1})^{c'^{-1}}$ as $\mathcal{H}(g^{f'} \cdot pk'^{-c'}) = \mathcal{H}(g^{f'} \cdot ((g^{f'} \cdot A'^{-1})^{c'^{-1}})^{-c'}) = \mathcal{H}(A') = c'$. The public key depends on the faked proof, i.e., the dishonest prover must compute the proof before choosing pk'.

Because of this attack, the use of the weak variant may invalidate the proof of knowledge. The authors of [BPW12] stated that the weak Fiat-Shamir transformation can safely be used when the statement (in this example, the public key) is fixed first (as in the attack the public key depends on the proof). However, since exponentiation Mix-Nets or re-encryption Mix-Nets are carried on as part of other bigger protocols, the list of inputs is not a-priori known. Moreover, it is interesting to investigate the impact of the weak variant in the context of Mix-Nets. Note also that the weak Fiat-Shamir transformation is widely used in practice when it comes to non-interactive proof systems: in [DMWG23] the authors examined over 75 different open-source implementations of proof systems that use the Fiat-Shamir heuristic and found 36 systems using the weak variant.

Remark:

We note that, according to [JJ01], and in the case of RSA-based decryption Mix-Nets, a ZKP may not be feasible; hence, defending against the attack described in Section 3.2.1 can be challenging. As suggested by Pfitzmann and Pfitzmann in [PP90], a public bulletin board should be avoided in this case and the senders should broadcast simultaneously their messages to the Mix-Nets in order to protect the users from this attack.

3.3 Refined Modeling of Cryptographic Primitives

In this section, we propose a refined modeling, in symbolic models, of certain cryptographic primitives that are based on modular exponentiation. Our objective is to reduce the gap to computational models while maintaining a high level of automation. We are particularly interested in modeling ElGamal encryption and signatures, and ZKPs of the discrete logarithm’s knowledge.

3.3.1 Refined Model of ElGamal and DSA Signatures

ElGamal asymmetric encryption is typically modeled the same way as any other public key encryption schemes: using a model for an abstract standard probabilistic asymmetric encryption scheme, as in Equation (2.1) (e.g., in [CS11, DJL14]). When formally analyzing the electronic exam protocol Remark!, Dreier et al. proposed, in [DGK+14], the following equations for modeling ElGamal encryption and signatures:

$$\begin{aligned} \mathsf{dec}(\mathsf{enc}(m, \mathsf{pseudopub}(\mathsf{pk}(k), rce), r), \mathsf{pseudopriv}(k, \mathsf{exp}(rce))) &= m \\ \mathsf{checksign}(\mathsf{sign}(m, \mathsf{pseudopriv}(k, \mathsf{exp}(rce))), \mathsf{pseudopub}(\mathsf{pk}(k), rce)) &= m \end{aligned} \tag{3.1}$$

However, these equations are linked to their model of exponentiation Mix-Nets and which hides all exponentiations using the functions used in the equations. They involve four constructors: pk, which takes as argument a secret key and computes the corresponding public key, exp, which takes as argument the exponentiation Mix-Net secret exponent and computes the new basis, a binary function pseudopub, taking as argument a public key and the Mix-Net exponent and a binary function pseudopriv, taking as argument a private key and a basis of the finite group. It is clear that the exponentiations are hidden away by the functions pseudopub and pseudopriv.

When modeling ElGamal encryption and signatures, we set the objective of the model to be independent of the protocol and taking into account the fact that public keys are based on the modular exponentiation of the form $\mathsf{exp}(\cdot, \cdot)$ and not $\mathsf{pk}(\cdot)$. An intuitive approach to model ElGamal encryption, where public keys are the result of the exponentiation operator, would be as follows: $\mathsf{dec}(\mathsf{enc}(m, \mathsf{exp}(g, sk), r), sk) = m$, where g is a fixed generator. However, this equation only allows encryption under public keys of the form $\mathsf{exp}(g, sk)$, whereas in the ElGamal cryptosystem one can encrypt under different bases. Another attempt at modeling ElGamal would be to replace the fixed generator g in the aforementioned equation with a variable to allow encryption with different bases: $\mathsf{dec}(\mathsf{enc}(m, \mathsf{exp}(X, sk), r), sk) = m$. This equation is however incorrect with regard to ElGamal encryption and decryption algorithms. Consider a dishonest party constructing their public key $D = \mathsf{exp}(H, d)$ from an honest key $H = \mathsf{exp}(g, h)$. When the dishonest party receives a message m encrypted with their public key $\mathsf{enc}(m, \mathsf{exp}(H, d), r)$, they should not be able to obtain m as when deciphering one obtains $m \cdot ((g^h)^d)^r \cdot ((g^r)^d)^{-1} \neq m$. Yet, with the latter equation we have $\mathsf{dec}(\mathsf{enc}(m, \mathsf{exp}(H, d), r), d) = m$. To avoid this issue, we include the generator used in the encryption in the equation, which is consistent with the real cryptographic primitive (the public key includes the generator as a parameter). To that end, we use the equations depicted in Table 3.1.

Primitive	**Equation**
Exponentiation	$\mathsf{exp}(\mathsf{exp}(g,x),y) = \mathsf{exp}(\mathsf{exp}(g,y),x)$ $\mathsf{exp}(\mathsf{exp}(\mathsf{exp}(g,x),y),z) = \mathsf{exp}(\mathsf{exp}(\mathsf{exp}(g,x),z),y)$
ElGamal Encryption	$\mathsf{dec}(\mathsf{enc}(m,X,\mathsf{exp}(X,s),r),X,s) = m$
ElGamal Signature	$\mathsf{checksign}(\mathsf{sign}(m,X,s),X,\mathsf{exp}(X,s)) = m$

Table 3.1: Equational Theory.

Our equations involve a single binary function exp which denotes the operation of modular exponentiation. The first equation for the exponentiation is the standard equation used for the protocol involving DH exponentiation in ProVerif. The second equation is needed, as we will see in Sections 3.4 and 3.5, to find the attacks against protocols involving exponentiation Mix-Nets. For example, the attack described in Section 3.2.3 needs three exponentiations. To encrypt a message m with ElGamal, one needs to generate a random r and to use the public parameters of the receiver: a basis X and the key $\mathsf{exp}(X,s)$ such that s is the secret key of the receiver. To sign a message m, one needs to specify the basis X used in the public parameter $\mathsf{exp}(X,s)$ and the secret exponent s. Then, $\mathsf{sign}(m,X,s)$ is a digital signature for the message m. To verify the digital signature, one needs the corresponding public key $\mathsf{exp}(X,s)$ and the basis X. If we do not assume the signature scheme to be hiding, we can add the equation $\mathsf{getmess}(\mathsf{sign}(m,X,s)) = m$ to the equational theory depicted in Table 3.1. This model of ElGamal probabilistic public key encryption is not bound to any protocol, and more precise than the usual models, as it inherits the algebraic properties of the exponentiation operation. Obviously, it can also be used in protocols that do not include exponentiation Mix-Nets. Our model of the ElGamal encryption together with the equations for the modular exponentiation have allowed us to find a subtle attack described later in Section 3.5.3.

3.3.2 Refined Model of Zero-Knowledge Proofs

We enriched our model with a non-interactive zero knowledge proof (NIZKP) of the discrete logarithm's knowledge since the sender is required to transmit its message along with a ZKP proving their knowledge of either the secret key for the exponentiation Mix-Nets or the randomness used for ElGamal encryption for the re-encryption Mix-Nets. Upon a valid verification of the proof given by the sender, the Mix-Net servers accept the entry. Otherwise, they reject the message.

We use two different models, one modeling a weak ZKP, and the other modeling a strong ZKP. The corresponding equations are depicted in Table 3.2.

Primitive	Equation
Strong ZKP	$\mathsf{ck}(\mathsf{szkp}(A, g, x), g, \mathsf{exp}(g, x), \mathsf{Hash}(g, \mathsf{exp}(g, x), A)) = \mathsf{true}$
Weak ZKP	$\mathsf{ck}(\mathsf{wzkp}(A, X, x), X, \mathsf{exp}(X, x), \mathsf{Hash}(A)) = \mathsf{true}$

Table 3.2: Equational Theory for Modeling ZKPs.

These equations state that the verification of a proof of knowledge of a secret key succeeds only if the proof was created genuinely: it needs to be checked using the same generator, public key and hash that was used to generate it. The first argument of both functions (szkp and wzkp) refers to the commitment used in the proof. The hash of the commitment is used by ck to check the adequacy of the challenge, and in the case of the strong variant the hash includes the public key. Moreover, in case of the strong ZKP, the proof is only valid when using the public generator g, as this is part of the statement. In the case of a weak ZKP, the hash only includes the commitment, and the generator is now a variable as it is no longer part of the statement.

3.4 Formal Model of Exponentiation and Re-Encryption Mix-Nets

We model Mix-Net protocols in the applied Π-Calculus described in Section 2.1 and perform the automatic protocol verification with ProVerif. Mix-Nets are supposed to establish anonymous channels to guarantee the privacy of the participants. We model privacy properties as equivalence properties. More precisely, honest parties are modeled as processes which can exchange messages on public or private channels, create keys or fresh random values, and perform tests and cryptographic operations. The attacker has complete control of the network, except for the private channels.

A Mix-Network, as explained in the latter sections, involves different parties, among which are the Mix-Net servers, the senders who send the messages to the Mix-Net servers, and the receivers of the mixed messages. Formally, without loss of generality, we define Mix-Nets as follows.

Definition 3.4.1 (Mix-Net). *A Mix-Net is a tuple* $(\mathsf{S}, \mathsf{M}, \mathsf{R}, \tilde{\mathsf{n}}_p)$ *where* S *is the process executed by the senders,* M *is the process executed by the Mix-Net servers,* R *is a process executed by the receivers, and* $\tilde{\mathsf{n}}_p$ *is a set of private channel names, where* R *can be the null, one or many processes.*

We note that all senders and all mixing servers execute the same process, but with different variable values, e.g., keys, randomness and identities. We give a formal specification of the Mix-Net protocol.

Definition 3.4.2 (Mix-Net instance). *A* Mix-Net instance *is a closed process* $\mathsf{MP} = \nu\tilde{\mathsf{n}} \cdot (\mathsf{S}\sigma_{\mathsf{id}_1}\sigma_{\mathsf{skS}_1}| \dots | \; \mathsf{S}\sigma_{\mathsf{id}_j}\sigma_{\mathsf{skS}_j} \; |\mathsf{M}\sigma_{\mathsf{m}_1}| \dots \; |\mathsf{M}\sigma_{\mathsf{m}_k} \mid \mathsf{R}\sigma_{\mathsf{id'}_1}\sigma_{\mathsf{skR}_1}| \dots | \; \mathsf{R}\sigma_{\mathsf{id'}_j}\sigma_{\mathsf{skR}_j}))$ *where* $\tilde{\mathsf{n}}$ *is the set of all restricted names, which includes the set of the protocol's private channels;* $\mathsf{S}\sigma_{\mathsf{id}_i}\sigma_{\mathsf{skS}_i}$ *are the processes run by the senders with the substitutions specifying the identity and the secret key of the* i^{th} *sender respectively;* $\mathsf{M}\sigma_{\mathsf{m}_j}$ *is the process executed by the* Mix-Net *servers with the substitution specifying in particular their secret shuffle variable, and* $\mathsf{R}\sigma_{\mathsf{id'}_i}\sigma_{\mathsf{skR}_i}$ *are the processes run by the receivers with the substitutions specifying the identity and the secret key of the* i^{th} *receiver respectively.*

Since both exponentiation and re-encryption Mix-Nets are meant to guarantee the *anonymity* of the senders, we define the *anonymous shuffling* property with regard to only the senders. We write $\mathsf{MP}_{\{\mathsf{id}_1,\mathsf{id}_2\}}[\cdot]$ for the Mix-Net instance MP without the processes for the senders id_1 and id_2.

Definition 3.4.3 (Anonymous Shuffling). *A* Mix-Net *ensures* Anonymous Shuffling *if for any mixnet processes* MP, *any senders* id_1 *and* id_2 *and any private materials* p_1 *and* p_2 *(i.e., their keys or messages):*
$\mathsf{MP}_{\{\mathsf{id}_1,\mathsf{id}_2\}}[\mathsf{S}\sigma_{\mathsf{id}_1}\sigma_{\mathsf{p}_1} \mid \mathsf{S}\sigma_{\mathsf{id}_2}\sigma_{\mathsf{p}_2}] \approx_l \mathsf{MP}_{\{\mathsf{id}_1,\mathsf{id}_2\}}[\mathsf{S}\sigma_{\mathsf{id}_1}\sigma_{\mathsf{p}_2} \mid \mathsf{S}\sigma_{\mathsf{id}_2}\sigma_{\mathsf{p}_1}]$

In the case of exponentiation Mix-Nets, this definition requires that the process where id_1 has sk_1 and id_2 has sk_2 as their secret key, is equivalent to the process where id_2 has sk_1 and id_1 has sk_2. This prevents the attacker from obtaining information about the identity of the sender based on the outcome of the Mix-Net. Likewise, in the case of re-encryption Mix-Nets the process where id_1 encrypts the message m_1 and id_2 encrypts the message m_2, should be equivalent to the process where id_2 encrypts m_1 and id_1 encrypts m_2. This prevents the attacker from having any information about which plaintext belongs to which sender. As stated above, we consider an attacker having complete control of the network and, depending on the properties, we also allow the attacker to corrupt parties. Such corrupted parties cooperate with the attacker by revealing their secret data and/or taking instructions from them. We model them using the definition given in [DKR06]: if the process P is an honest party, then the process P^{c_1,c_2} is its corrupted version. The latter is a variant of the honest process sharing with the attacker channels c_1 and c_2. P^{c_1,c_2} sends all its inputs and freshly generated names to the attacker via channel c_1, and receives (via channel c_2) messages from the attacker that will determine its behavior.

Naturally, it is interesting to consider corrupted or dishonest senders, as they might be interested in obtaining a link between the outcome of a Mix-Net and the identity of some other users. We can do this by replacing honest senders with corrupted ones. For example, if we assume that a sender id_3 is dishonest, we obtain:

$$\begin{aligned}MP_{\{\mathsf{id}_1,\mathsf{id}_2,\mathsf{id}_3\}}[S\sigma_{\mathsf{id}_1}\sigma_{p_1} \mid S\sigma_{\mathsf{id}_2}\sigma_{p_2} \mid (S\sigma_{\mathsf{id}_3}\sigma_{p_3})^{c_1,c_2}] \approx_l \\ MP_{\{\mathsf{id}_1,\mathsf{id}_2,p_3\}}[S\sigma_{\mathsf{id}_1}\sigma_{p_2} \mid S\sigma_{\mathsf{id}_2}\sigma_{p_1} \mid (S\sigma_{\mathsf{id}_3}\sigma_{p_3})^{c_1,c_2}]\end{aligned}$$

Remark:

Some researchers may refer to *unlinkability* or *untraceability* when analyzing Mix-Nets. For example, the authors of [RGL22] describe their attack against exponentiation MIX-NET as a "*unlinkability attack*". We argue that anonymous shuffling is not an unlinkability property as defined informally by the ISO/IEC standard 15408 [ISO09].

3.4.1 Formal Analysis of Exponentiation MIX-NETS

As explained in Section 3.3.1, exponentiation MIX-NETS have been modeled previously, but only as part of other, bigger protocols, for example in the analysis of the E-exam protocol Remark! [DGK+14]. This modeling is actually sufficient to represent the MIX-NET functionality, but it is clearly insufficient to capture the attack aginst exponentiation MIX-NETS described in Section 3.2.3.

Model. The process for each MIX-NET server M is given in Figure 3.4, the honest sender process is: `let S(skS) = out(ch, (exp(g, skS)).` The sender simply outputs their public key. Each MIX-NET server inputs the list of all keys, checks that they are distinct, applies their random exponent to all keys and outputs the list of all keys in random order (modeled using parallel outputs on the same channel). We note that all keys should be different. Otherwise a trivial attack consists in copying the public key to track: the corresponding pseudonym is the one figuring twice in the list output by the MIX-NET.

Equational theory. We use the equations for the exponentiation and ZKPs depicted in Table 3.1. We note that in the equations g is a fixed generator and not a variable, as otherwise the pre-treatment of equations by PROVERIF does not terminate. The first equation for exponentiation takes into account the equation needed for the protocol to work but is insufficient to capture attacks against exponentiation MIX-NETS. This is the reason why we need to explicitly give an equation for three exponents, as the attack uses an additional exponentiation. Adding additional exponentiation (more than three) to the equational theory caused non-termination of PROVERIF in our examples.

In PROVERIF, we modeled only one MIX-NET server and add dedicated private channels between the two participants swapping their keys and the MIX-NET

```
let M($e_{N_j}$) =
   in(ch, $pk_{S_1}$).
      ⋮
   in(ch, $pk_{S_k}$).
   if ($pk_{S_1}$ <> $pk_{S_2}$)&& ... &&($pk_{S_1}$ <> $pk_{S_k}$) then
      ⋮
   if ($pk_{S_k}$ <> $pk_{S_1}$)&& ... &&($pk_{S_k}$ <> $pk_{S_{k-1}}$) then
   (out(ch, exp($pk_{S_1}$,$e_{N_j}$)) ||
      ⋮
    out(ch, exp($pk_{S_1}$,$e_{N_j}$))).
```

Figure 3.4: Mix-Net's Process.

to ensure that both keys take part in the mixing. The keys are still published, and the attacker can still add a malicious key to the mixing.

Analysis. The result of the analysis of exponentiation Mix-Nets is given in Table 3.4. ProVerif concludes that *Anonymous Shuffling* property is not satisfied and finds the exact same attack depicted in Figure 3.3 when ZKPs are absents from the model. When using the weak ZKP, ProVerif returns the same attack trace on the protocol. We note that in our ProVerif code, when integrating the ZKPs and when the Mix-Net verifies a weak proof, we input the value X from the network – fixing g would prevent the attacker from using a public of the form $\mathsf{exp}(\mathsf{exp}(g, x), a)$, as in the first attack trace. With the strong variant ProVerif concludes that anonymous shuffling is satisfied (Table 3.4). The ProVerif attack with weak ZKP works in the same way as the one without a ZKP (Figure 3.3): an attacker builds a new key $pk' = pk^{\alpha}$ from their victim's key pk and bypasses the proof check since $ck(wzkp(A, pk, \alpha), pk, exp(pk, \alpha), h(A)) = true$. In reality, this attack does not work for any α, but for example choosing $\alpha = c'^{-1}$ results in a real attack as follows. Let $(c, f) = (\mathcal{H}(g^a), a + c \cdot sk)$ be the proof generated by an honest participant with public key $pk = g^{sk}$. An attacker computes $c' = \mathcal{H}(A^{c^{-1}})$, where $A = g^a$ is the commitment of the honest participant, chooses $pk' = pk^{c'^{-1}}$ as public key and computes $f' = c^{-1} \cdot f$. The attacker sends (c', f') as a proof along with their public key pk' to the Mix-Net. The verification procedure of the ZKP by the Mix-Net succeeds since we have $\mathcal{H}(g^{f'} \cdot (pk'^{c'})^{-1}) = \mathcal{H}(g^{c^{-1} \cdot a + sk} \cdot g^{-sk}) = c'$. As the verification of the proof succeeds, the attacker can then perform the same attack as described in Figure 3.3 by raising pseudonyms to the power of c'^{-1} (instead of a).

3.4.2 Formal Analysis of Re-Encryption Mix-Nets

In the following, we conduct a formal analysis of re-encryption Mix-Nets.

Model. The process for each MIX-NET server M is similar to the one defined for exponentiation MIX-NETS (as depicted in Figure 3.4), but it uses different parameters. Each MIX-NET server inputs the list of all generated ciphertexts, checks if they are distinct, applies a random coin to each ciphertext in order to re-randomize it and outputs the list of all new ciphertexts in random order.

The sender simply outputs its ciphertext. Thus, the sender process is: let S(m_S, r_S, pk) = out(ch, (enc(m_S, g, pk, r_S)).

Equational theory. Re-encryption MIX-NETS rely on the homomorphic properties of the encryption scheme since MIX-NET servers need to re-encrypt the list of the inputted ciphertexts. Therefore, in addition to the encryption equation, we require an additional equation that models a re-encryption operation. We use the equational theory depicted in Table 3.3

Primitive	**Equation**
Encryption	$\mathsf{dec}(\mathsf{enc}(m, X, \mathsf{exp}(X, x), r), X, x) = m$
Re-Encryption	$\mathsf{reenc}(\mathsf{enc}(m, X, \mathsf{exp}(X, x), r), r', X, \mathsf{exp}(X, x)) = \mathsf{enc}(m, X, \mathsf{exp}(X, x), \mathsf{sum}(r, r'))$
Exponentiation	$\mathsf{EXP}(\mathsf{enc}(m, X, \mathsf{exp}(X, x), r), a) = \mathsf{enc}(\mathsf{exp}(m, a), X, \mathsf{exp}(X, x), \mathsf{mult}(r, a))$

Table 3.3: Equational Theory for Modeling Re-Encryption MIX-NET.

The first equation corresponds to our refined model of ELGAMAL described in Section 3.3.1. Let $(c_1 = g^r, c_2 = m \cdot (g^x)^r)$ be an ELGAMAL ciphertext. Re-encrypting the latter ciphertext consists in generating a new random r' and multiplying c_1 with $g^{r'}$, and c_2 with $(g^x)^{r'}$. The resulting ciphertext is of the form $(g^{r+r'}, m \cdot (g^x)^{r+r'})$. To model the re-encryption operation, we introduce two function symbols sum and reenc. The binary function sum represents a sum of two exponents (albeit in a limited way, as sum has no further equations – ideally sum should be associative and commutative, but this is not possible in PROVERIF). The function reenc takes as arguments a ciphertext $\mathsf{enc}(m, X, \mathsf{exp}(X, x), r)$, a coin r', a basis X and a public key $\mathsf{exp}(X, x)$. It returns the ciphertext $\mathsf{enc}(m, X, \mathsf{exp}(X, x), \mathsf{sum}(r, r'))$ with a coin representing the sum of r and r'. The first and the second equations are needed for the protocol to work as it uses encryption and re-encryption, but they are insufficient to capture attacks against re-encryption MIX-NETS. To capture the attack depicted in Figure 3.2, we need to augment our equational theory with an equation that allows for the exponentiation of the ciphertext with respect to the induced homomorphic properties on the corresponding plaintext. The third equation uses the binary function EXP (different from exp) to apply a known exponent a over the

Protocol	ZKP	Result	Time
Exponentiation MIX-NETS	without	✗	2 s
	weak	✗	1 m 6 s
	strong	✓	3 s
Re-encryption MIX-NETS	without	✗	1 s
	weak	✗	2 s
	strong	✓	1 s

Table 3.4: Results of our analysis of Anonymous Shuffling, with and without the added ZKP.

plaintext. We introduce the function mult to represent a multiplication between random coins (again, in a limited way). When analyzing re-encryption MIX-NETS using ZKPS to show knowledge of the randomness used for the encryption, we also added the equations depicted in Table 3.2.

Analysis. The result of the analysis of re-encryption MIX-NETS is given in Table 3.4. PROVERIF concludes that *Anonymous Shuffling* is not satisfied and finds the attack depicted in Figure 3.2 when the protocol is modeled without ZKPS. In the case of the weak ZKP, PROVERIF finds (nearly) the same attack, except that the attacker needs to fake the ZKP as in the attack on the exponentiation MIX-NET with weak ZKP. In contrast, the property is verified when the strong ZKP is used instead.

Remark:

We note that the attacks found by PROVERIF would not be possible if we modeled a dishonest user the standard way: we create a secret key (or a secret message) for the dishonest party and we output it to the attacker on the public channel. In order to find the attacks described above, the secret keys (or the messages) of the dishonest user need to be generated dishonestly which we model as an input from the public channel.

3.5 Applications

In this section, we perform formal analysis of four cryptographic protocols using MIX-NETS for different privacy purposes. A formal analysis of the electronic exam protocol has been presented in [DGK$^+$14] where the protocol has been proved secure and no attacks have been found against privacy properties. Regarding the three other protocols, to the best of our knowledge, we present their first formal analysis. The timings for the results of the formal analysis given below were done on macOS with a M1 processor and 16 GB of RAM. All of our ProVerif files are publicly accessible and can be accessed online through the Gitlab repository[1]. We use a public Bulletin Board, denoted **BB**. A Bulletin Board is a public append-only (*i.e.*, nobody can delete) broadcast message channel. We also assume that the exponentiation and re-encryption Mix-Nets behave correctly. For simplicity, we have modeled the Mix-Nets performed by all servers as a single honest Mix-Net server. All the aforementioned protocols are analyzed with and without the proposed fix described in Section 3.4.1.

3.5.1 Remark! Protocol

We first analyze the Remark! protocol, an electronic exam protocol, designed by Giustolisi *et. al* [GLR14] and meant to achieve multiple privacy properties without relying on trusted parties. The following description of the protocol is based on [GLR14, DGK$^+$14].

Protocol Description. The Remark! protocol involves four types of parties: Mix-Net servers, the candidate(s) sitting the exam, the examiner(s) correcting the answers and marking them, and an exam authority collecting the answers, dispatching them for the marking phase and delivering the final marks. It is assumed that each party is given a pair of public/private ElGamal keys with a common generator g, *i.e.,* the private key x and the public key $y = g^x$. The protocol runs in four phases: Registration, Examination, Marking and Notification. The protocol's sequence diagram is depicted in Figure 3.5.

Registration. The registration phase uses an exponentiation Mix-Net to generate pseudonyms $\overline{pk_C}$ and $\overline{pk_E}$ for both candidates C and examiners E based on their public keys pk_C and pk_E as described in Section 3.2.3. Let r_m and r'_m denote the Mix-Net's exponents related to candidates and examiners respectively, and let ($h_C = g^{r_m}$, $h_E = g^{r'_m}$) be the new generators.

Examination. The exam authority begins by signing (using its secret key sk_A) and encrypting (with the candidates' pseudonyms $\overline{pk_C}$) the questions q and publishes the result on the bulletin board **BB**. Then, the exam authority collects the candidates' answers a (which are signed with the candidates' pseudonym keys, and encrypted with the authority's key pk_A), verifies the signatures, resigns them, encrypts them using the corresponding candidates' pseudonyms and publishes them.

Marking. The exam authority encrypts the answers with examiners pseudonyms $\overline{pk_E}$ and publishes them. Each examiner marks the received tests with a mark $mark$, signs them using its pseudonym, encrypts them with the pkA and sends the marked tests back to the authority.

[1] `https://gitlab.limos.fr/dhmahmoud/usenix24-632/`

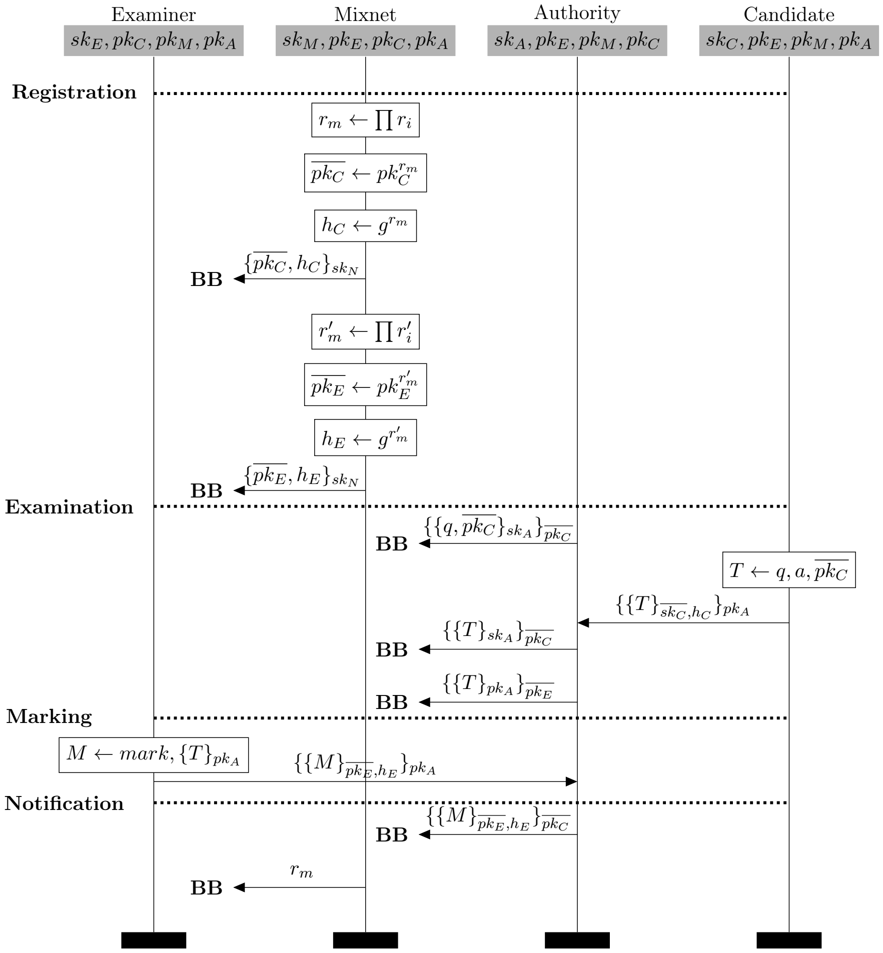

Examiner
sk_E, pk_C, pk_M, pk_A
Mixnet
sk_M, pk_E, pk_C, pk_A
Authority
sk_A, pk_E, pk_M, pk_C
Candidate
sk_C, pk_E, pk_M, pk_A
Registration
$r_m \leftarrow \prod r_i$
$\overline{pk_C} \leftarrow pk_C^{r_m}$
$h_C \leftarrow g^{r_m}$
$\{\overline{pk_C}, h_C\}_{sk_N}$
BB
$r'_m \leftarrow \prod r'_i$
$\overline{pk_E} \leftarrow pk_E^{r'_m}$
$h_E \leftarrow g^{r'_m}$
$\{\overline{pk_E}, h_E\}_{sk_N}$
BB
Examination
$\{\{q, \overline{pk_C}\}_{sk_A}\}_{\overline{pk_C}}$
BB
$T \leftarrow q, a, \overline{pk_C}$
$\{\{T\}_{\overline{sk_C}, h_C}\}_{pk_A}$
$\{\{T\}_{sk_A}\}_{\overline{pk_C}}$
BB
$\{\{T\}_{pk_A}\}_{\overline{pk_E}}$
BB
Marking
$M \leftarrow mark, \{T\}_{pk_A}$
$\{\{M\}_{\overline{pk_E}, h_E}\}_{pk_A}$
Notification
$\{\{M\}_{\overline{pk_E}, h_E}\}_{\overline{pk_C}}$
BB
r_m
BB

Figure 3.5: Remark! protocol diagram.

Notification. The authority receives the marks, verifies the signatures of the examiners and publishes the signed marks encrypted with the candidates' pseudonyms. Finally, the MIX-NET servers de-anonymize the candidates pseudonyms by revealing the secret exponent r_m.

Formal Analysis. Dreier *et. al* [DGK$^+$14] formalized privacy properties for electronic exams. Here we focus on *Anonymous Marking* and *Anonymous Examiner*'s informal definitions. An e-exam protocol ensures *Anonymous Marking* if a process where two candidates C_1 and C_2 answer a_1 and a_2 respectively, is indistinguishable from a process where the candidates switch their answers. This means that an examiner or an attacker cannot link a candidate to a certain answer which ensures impartiality. Similarly, it ensures *Anonymous Examiner* if a process where two examiners E_1 and E_2 grade exam forms f_1 and f_2 respectively, is indistinguishable from a process where the examiners switch the forms. This property prevents the attacker and the candidates from obtaining information about the identity of the examiner who marked a certain answer to prevent coercion.

The mentioned properties were proven satisfied with ProVerif on the model given in [DGK$^+$14]. This model uses the abstract model for the Mix-Net described in the beginning of Section 3.4.1. We took the existing ProVerif files and replaced the abstract model of the Mix-Net with our refined model, and checked the same properties. Now, ProVerif finds attacks on Anonymous Examiner and Anonymous Marking. The results of the analysis are depicted in Table 3.5.

Anonymous Marking is not satisfied because a candidate C_i signs his answer a_i with his pseudonym $\overline{pk_{C_i}}$ and the pseudonym generated by the exponentiation Mix-Net can be linked back to the candidate using the attack described in Section 3.2.3. In particular, a dishonest examiner can infer the author of the copy of the exam he is correcting. This breaks the desired candidates' privacy and the expected fairness of the marking procedure. Similarly, Anonymous Examiner is not satisfied because the examiners sign the marks they attribute using their pseudonyms, and again the pseudonyms generated by the exponentiation Mix-Net are linkable. A dishonest party can perform the same attack to track the examiner's pseudonym and to learn which examiner has corrected a given copy. This breaks the desired examiners' privacy property and might make them vulnerable to coercion.

The results of the analysis are depicted in Table 3.5.

3.5.2 Crypto Santa Protocol

The second protocol we have analyzed is the Crypto Santa protocol. It is based on a Christmas tradition called Secret Santa. During this ceremony, the participants are randomly assigned a person to whom they give a gift. In particular, the identity of the gift giver is to remain a secret. The Crypto Santa protocol is a cryptographic protocol designed by P. Ryan [Rya15] to cryptographically implement the Secret Santa tradition. The protocol is designed to protect the anonymity of the gift giver, and is essentially based on an Exponentiation Mix-Net.

Protocol Description. The participants are n players. Each player is assumed to have a pair of a public and secret keys (pk_i, sk_i) where $pk_i = g^{sk_i}$. The public keys are arranged into a list $\tilde{L} = (pk_1, \ldots, pk_n)$ such that the key pk_i belongs to the player P_i. Given a list L let $L[j]$ denote the j^{th} term of the list L. The players take the list of public keys in turns, and each player i performs an exponentiation with s_i and a shuffling step.

The final output is the list $L_n = \Pi_n(L_{n-1}[1]^{s_n}, \ldots, L_{n-1}[n]^{s_n})$ along with the final generator $g^s = g^{s_1 s_2 \cdots s_n}$. Note that the position of each pseudonym in the final outputted list L_n is relevant, as it determines a player to whom the gift is offered. If $L_n[j] = pk_i^s$, then P_i presents their gift $gift_i$ to the player P_j.

Protocol	ZKP	Property	Result	Time
Remark! [GLR14]	without	Anonymous Marking	✗	3 m 16 s
		Anonymous Examiner	✗	4 m 19 s
	weak	Anonymous Marking	✗	9 m 35 s
		Anonymous Examiner	✗	9 m 23 s
	strong	Anonymous Marking	✓	11 s
		Anonymous Examiner	✓	7 s
Haenni Voting [HS11]	without	Vote Privacy	✗	4 m 35 s
	weak		✗	9 m 35 s
	strong		✓	14 s
Crypto Santa [Rya15]	weak	Anonymous Shuffling	✗	4 m 6 s
	strong		✓	9 s
IVXV [IVX23]	without	Vote Privacy	✗	1 s
	weak		✗	25 s
	strong		✓	8 s

Table 3.5: Results of our analysis, with and without the added ZKP. Crypto Santa required a ZKP from the start.

To prevent a player from cheating, the protocol suggests that each player provides two different Zero-Knowledge Proofs: a proof that he has performed the shuffle correctly (based on the specifications from [Wik05]), and a proof of knowledge of their secret key sk_i using a standard ZK proof of the discrete log [Rya15]. We do not give further details about the proof of shuffle since in our model we assume that the shuffle procedure is performed correctly.

Formal Analysis. In [Rya15] there is no formal definition of the security properties ensured by Crypto Santa protocol. Nevertheless, they emphasized the fact that the gift giver should be anonymous. Hence, in the final list, the players' pseudonyms should be anonymized. Thus, the Crypto Santa protocol should probably guarantee Anonymous Shuffling (Def. 3.4.3). The results of our analysis are depicted in Table 3.5: Anonymous Shuffling is satisfied in case of strong ZKPs, but broken in case of weak ZKPs. Note that the original paper [Rya15] does not specify which ZKP is to be used, and weak ZKPs are widely used [DMWG23].

3.5.3 Haenni's Internet Vote Protocol

The next protocol we analyze is an internet vote protocol proposed in [HS11]. This paper also introduced the concept of exponentiation Mix-Nets. The se-

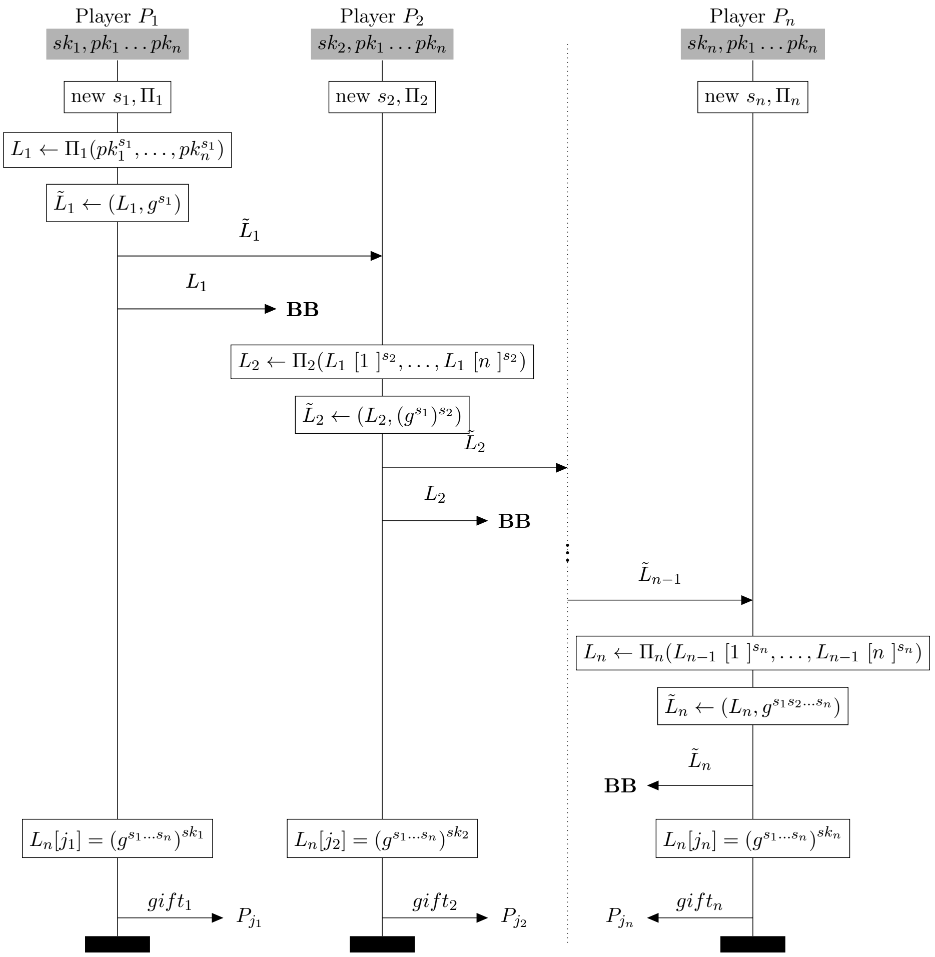

Player P_1
$sk_1, pk_1 \dots pk_n$
Player P_2
$sk_2, pk_1 \dots pk_n$
Player P_n
$sk_n, pk_1 \dots pk_n$
new s_1, Π_1
new s_2, Π_2
new s_n, Π_n
$L_1 \leftarrow \Pi_1(pk_1^{s_1}, \dots, pk_n^{s_1})$
$\tilde{L}_1 \leftarrow (L_1, g^{s_1})$
$\tilde{L}_1$
L_1
BB
$L_2 \leftarrow \Pi_2(L_1\ [1\ ]^{s_2}, \dots, L_1\ [n\ ]^{s_2})$
$\tilde{L}_2 \leftarrow (L_2, (g^{s_1})^{s_2})$
$\tilde{L}_2$
L_2
BB
$\tilde{L}_{n-1}$
$L_n \leftarrow \Pi_n(L_{n-1}\ [1\ ]^{s_n}, \dots, L_{n-1}\ [n\ ]^{s_n})$
$\tilde{L}_n \leftarrow (L_n, g^{s_1 s_2 \dots s_n})$
$\tilde{L}_n$
BB
$L_n[j_1] = (g^{s_1 \dots s_n})^{sk_1}$
$L_n[j_2] = (g^{s_1 \dots s_n})^{sk_2}$
$L_n[j_n] = (g^{s_1 \dots s_n})^{sk_n}$
$gift_1$
P_{j_1}
$gift_2$
P_{j_2}
P_{j_n}
$gift_n$

Figure 3.6: Crypto Santa Protocol

curity of this voting protocol relies on the anonymity obtained by shuffling the voters' public keys. Casting a vote consists in signing the encrypted candidate choice with the anonymized public key generated by the MIX-NET. To obtain the election result, votes carrying a valid signature are decrypted and counted.

Protocol Description. This voting protocol involves four types of parties: the anonymizers, which are the MIX-NET servers, the voters, the talliers, and an election authority. The protocol runs in four phases: Registration, Preparation, Vote Casting and Tallying. We assume an anonymous channel $\mathcal{C}$ between the voters and the Bulletin Board. The protocol's sequence diagram is depicted in Figure 3.7.

Registration. The election authority begins by setting up a Public Key Infrastructure for the voters. Each voter id_{V_i} is therefore equipped with a key pair $(sk_i, pk_i = g^{sk_i})$ and a public certificate binding their public key pk_i to their identity id_{V_i}.

Election Preparation. During this phase the election authority publishes the set C of possible candidates. The authority has to publish all certificates corre-

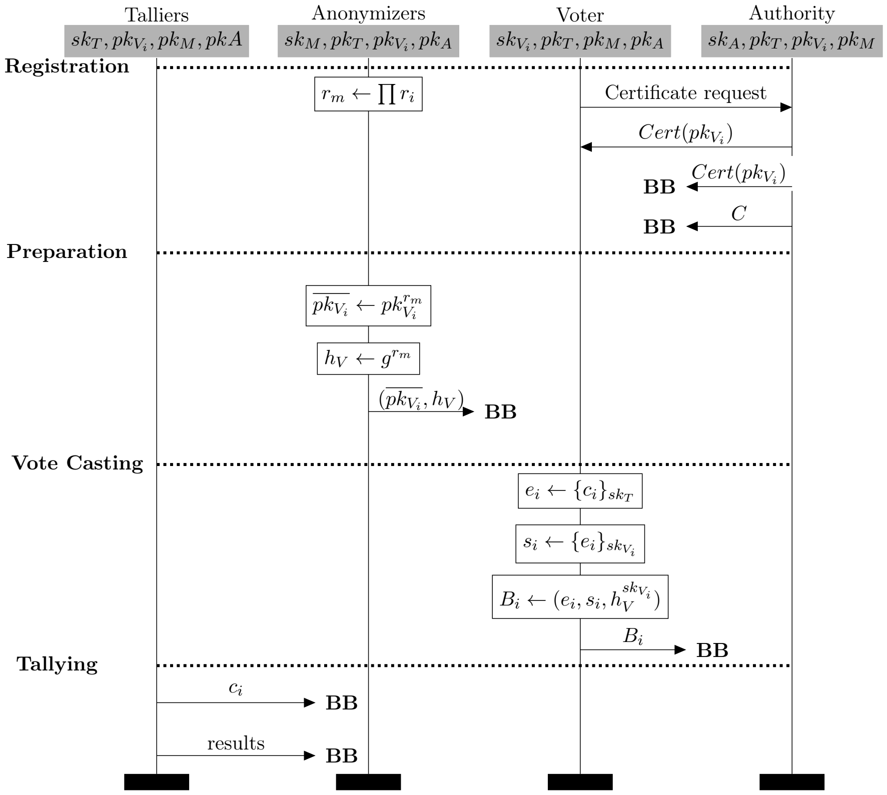

Talliers
Anonymizers
Voter
Authority
$sk_T, pk_{V_i}, pk_M, pkA$
$sk_M, pk_T, pk_{V_i}, pk_A$
$sk_{V_i}, pk_T, pk_M, pk_A$
$sk_A, pk_T, pk_{V_i}, pk_M$
Registration
$r_m \leftarrow \prod r_i$
Certificate request
$Cert(pk_{V_i})$
BB
$Cert(pk_{V_i})$
BB
C
Preparation
$\overline{pk_{V_i}} \leftarrow pk_{V_i}^{r_m}$
$h_V \leftarrow g^{r_m}$
$(\overline{pk_{V_i}}, h_V)$
BB
Vote Casting
$e_i \leftarrow \{c_i\}_{sk_T}$
$s_i \leftarrow \{e_i\}_{sk_{V_i}}$
$B_i \leftarrow (e_i, s_i, h_V^{sk_{V_i}})$
B_i
BB
Tallying
c_i
BB
results
BB

Figure 3.7: Haenni voting protocol sequence diagram.

sponding to eligible voters. Let $Y = \{y_1, \ldots, y_n\}$ be the list of voters' public keys. The anonymizers take as input Y and output the shuffled list $\tilde{Y} = \{\tilde{y_1}, \ldots, \tilde{y_n}\}$ together with the new generator $\tilde{g}$. The talliers jointly generate a public key using a threshold encryption scheme. The talliers jointly generate a public key using a threshold encryption scheme.

Vote Casting. Let $c_i \in C$ be an eligible candidate. To vote for c_i, a voter v_i needs to encrypt c_i with the talliers' public key y, signs the encrypted vote e_i with their private key sk_i, computes their pseudonym based on the new generator $\tilde{g}$ created by the anonymizers, and submits the ballot $B_i = (e_i, s_i, \tilde{g_i}^{sk_i})$ such that s_i corresponds to the encrypted and signed vote.

Tallying. Let $B = (e, s, \tilde{y})$ be a ballot. For a vote to be considered in the tally, the following conditions have to be satisfied: $\tilde{y}$ is a valid pseudonym, s is a valid signature for e and B is the only valid entry for $\tilde{y}$ in **BB**. The talliers decrypt all valid votes individually, and determine the final election result by applying the counting function to the resulting plaintexts. They also provide proofs of correct decryption.

Formal Analysis. The main security property related to vote protocols is *Vote Privacy.* We informally recall the definition proposed by Delaune *et. al* [DKR09]. In a nutshell, considering two voters V_1 and V_2 and their votes c_1 and c_2, respectively, a voting protocol respects vote privacy whenever a process where V_1 votes c_1 and V_2 votes c_2 is observationally equivalent to a process where V_1 votes c_2 and V_2 votes c_1. This means that an attacker is not able to detect whether arbitrary honest voters V_1 and V_2 swapped their votes or not. The result for Vote Privacy is depicted in Table 3.5. Once again, in the absence or in the presence of weak ZKPs, there is an attack; in the presence of strong ZKPs, the property is verified.

The attack found by ProVerif linking the pseudonyms generated by Mix-Net (which is used by the voters to sign their votes) and the identities of the voters is quite interesting as it also relies on our refined model of ElGamal encryption and is slightly different from the attack described in Section 3.2.3. More precisely, let us consider two voters V_i and V_j with secret keys skV_i and skV_j with corresponding public keys $pkV_i = g^{skV_i}$ and $pkV_j = g^{skV_j}$ respectively. An attacker who wants to track V_i would choose as public key ${pkV_i}^s$. The outputted list of the pseudonyms generated by the Mix-Net contains ${pkV_i}^{r_m}$ (the pseudonym of V_i), ${pkV_j}^{r_m}$ (the pseudonym of V_j) and ${pkV_i}^{sr_m}$, the attacker's pseudonym. To test whether a pseudonym $\tilde{g}$ belongs to the voter V_i, the attacker generates a plaintext m and encrypts it using their pseudonym h as the public key and using $\tilde{g}$ as the basis. If the attacker succeeds in decrypting the ciphertext using their secret key, then the pseudonym used as the basis for ElGamal encryption corresponds to V_i's pseudonym. We can see as follows why this attack works. Let c be the ciphertext for the plaintext m. Then, $c = (c_1, c_2)$ such that $c_1 = \tilde{g}^r = ({pkV_k}^{r_m})^r$ and $c_2 = m \cdot h^r = m \cdot ({pkV_i}^{sr_m})^r$. Using their secret key s the attacker tries to decrypt c. If $k = i$ then $c_2 \cdot c_1^{-s} = m$. ProVerif was only able to find such an attack because of our refined model of ElGamal encryption.

3.5.4 IVXV Internet Vote Protocol

The Estonian internet vote protocol uses the re-encryption Mix-Net to guarantee privacy of voters [IVX23]. It is used to conduct legally binding political elections in Estonia. We use the equational theory described in Table (3.3). The description of the protocol is based on the one given by Müller in [Mül22].

Protocol Description. The Voting System consists of the Election Organizer EO, the Vote Collector VC, the Registration Service RS, the I-Ballot Box Processor $IBBP$, the Talliers T and Mix-Nets MS. The protocol runs in four phases: *Registration*, *Vote Casting*, *Preparation* and *Tabulation*.

Registration. This scheme requires a Public Key Infrastructure. Each eligible voter v_i is therefore equipped with a key pair (sk_{v_i}, pk_{v_i}) and a public certificate $Cert_{v_i}$ binding their public key pk_{v_i} to their identity v_i. The EO publishes the set $C = \langle c_j \rangle$ of possible candidates. The talliers T jointly generate a public key pk_T using a threshold encryption scheme.

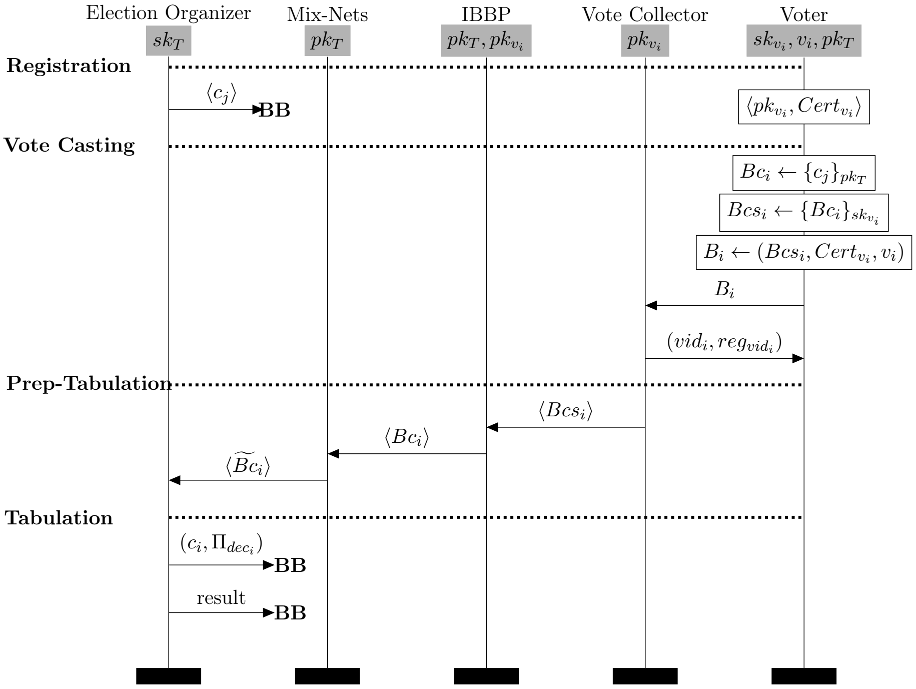


Figure 3.8: Estonian voting protocol sequence diagram.

Vote Casting. To vote for a candidate c_j, a voter v_i encrypts first c_j with the Talliers' public key pk_T and a generated coin rv_i. The voter signs the encrypted ballot Bc_i with their private key skv_i and sends their signed encrypted ballot Bcs_i along with their certificate $cert_{v_i}$ and their identifier v_i to VC. Upon receiving a valid $(Bcs_i, cert_{v_i}, v_i)$, VC responds with a unique generated identifier vid_i and the RS confirmation reg_{vid_i}.

Preparing for Tabulation. After the online voting phase, the VC has a set of digitally signed votes and RS has the set of registration queries and responses. Both of these sets are transferred to the $IBBP$ responsible for auditing the voting phase and pre-processing the votes for tabulation. $IBBP$ composes a new list of signed vote envelopes. This list only contains the latest vote for each voter v_i since the protocol allows voters to re-vote. $IBBP$ then extracts only the encrypted ballots from the list. $IBBP$ can then pass the new list $\langle Bc_i \rangle$ to the re-encryption Mix-Nets MS. The output set of the Mix-Nets $\langle \widetilde{Bc_i} \rangle$ is then sent to EO for tabulation.

Tabulation. The EO uses the election private key to decrypt each choice c_i and to compute the result. EO also provides a proof of correct decryption Π_{dec_i} for every plaintext.

Formal Analysis. We analyze *Vote Privacy* as described in the previous protocol analysis. The input list to the Mix-Nets in IVXV protocol is not public, but is given to a third party performing audits. The re-encryption Mix-Net was added to the protocol to preserve the privacy of voters with respect to this third party performing audits. Hence, vote privacy in the IVXV protocol relies on the anonymity provided by the shuffling procedure performed by the re-encryption Mix-Net.

We verify vote privacy using our new refined model of the Mix-Net. The result of the analysis is depicted in Table 3.5: ProVerif finds an attack on Vote Privacy. Assuming a malicious third party, the attacker can retrieve the encrypted ballot Bc_i of his victim v_i from the ballot list and perform the attack described in Section 3.2.2. At the end of the election, when the results are published, the attacker knows to which candidate c_j the voter v_i voted. In [Mül22], Müller found that the IVXV protocol is vulnerable to similar attacks exploiting the malleability of the ElGamal encryption when investigating the protocol manually. To fix this issue, he added ZKPs proving that the voter knows the randomness and the plaintext of the encrypted ballot. We also verified the protocol with an added weak or strong ZKP showing knowledge of the randomness (only), using the modeling described in Section 3.3.2. In the case of a weak ZKP the attack persists, whereas with a strong ZKP ProVerif succeeds in proving the property (Table 3.5).

3.6 Conclusion

In this work we propose a novel and more precise modeling of exponentiation Mix-Nets which includes the details of the exponentiation. Previous models used a high-level abstraction of the functionality, which lead them to miss attacks based on a weakness where a user can submit a modified version of a key of another participant in an attempt to trace them. Using three case studies (including a voting and an exam protocol) we show that we can use our improved modeling to analyze protocols using exponentiation Mix-Nets. In particular, we are able to (re-)discover known and unknown attacks on these protocols. Fixing these attacks requires the use of zero-knowledge proofs. We propose two models: a novel model for weak ZKPs vulnerable to certain attacks, and a model for strong ZKPs. We can show automatically that in our examples the use of weak ZKPs is insufficient, as all attacks persist. The use of strong ZKPs however fixes the attacks, and we can verify the protocols.

Moreover, we propose a novel and more precise modeling of ElGamal encryption where keys are actually the result of exponentiation operations, which is of independent interest and can be applied to other protocols, even if they do not use Mix-Nets. We also propose a refined modeling of re-encryption Mix-Nets, which allows use to rediscover a known attack on the Estonian voting protocol. Again, we can automatically show that the attack persists when adding weak ZKPs, but disappears if strong ZKPs are used.

As future work, we would like to apply our ElGamal and ZKP models to other protocols using these primitives, such as other voting protocols, private set intersection protocols or password based authentication protocols. Essentially, all existing ProVerif models of protocols using these primitives could be extended using our new refined modeling. We hope that in the future such a modeling will become commonplace. As previously stated, our full equational theory is not supported by the majority of the existing tools. For instance, although Tamarin's builtin model for DH exponentiation is the most complete (exponentiation is modeled as an associative-commutative function symbol), the tool cannot handle the exponentiation operator inside a user-defined equation. Further research is required to understand precisely how the tool can be adapted to support such equations. In contrast, the standard equation used for analyzing security protocols that use DH exponentiation in ProVerif pertains to modeling the minimal requirements for protocols to function (corresponding to the first equation of Table 3.1). For example, the recent framework Sapic^{+} [CJKK22] traduces the Tamarin's DH builtin into the aforementioned equation when automatically translating it into a ProVerif file. We show that adding an equation for triple exponentiation to the standard DH theory is rather simple yet greatly empowers the attacker by providing them with a degree of freedom over the exponentiation operator, potentially enabling certain attacks.

Moreover, we would like to get rid of some of the remaining limitations of the current model in ProVerif. For example, in the current equational theories, we use a fixed generator, which in particular limits the number of exponentiations, and some attacks need more than three exponentiations [PQ01]. In general, there are results showing that a fixed number can be sufficient (e.g., [Möd12]), which are however not directly applicable in this case. We would also like to propose models capturing other attacks on weak zero-knowledge proofs. Using our model, we were able to instantiate all attack traces found by the tool with real attacks. In fact, ProVerif cannot find the real attack directly as our equational theory does not encompass inverses (and adding them is currently not possible in ProVerif). However, our equations do not induce false vulnerabilities in the modeled cryptographic primitives since they represent a real (albeit abstracted) behavior of the primitive, and the missing values are simple to instantiate.

Finally, we do not consider the scenario of corrupted Mix-Nets. To model one or more corrupted Mix-Nets, additional details have to be considered within the model. The assumption of having at least one single honest server among the ones in Mix-Nets is sufficient to have an honest mixing process, as stated for example in [GLR14]. However, this assumption is not always true, as shown in [Wik04a]. The fifth and the second attack required that the first mix-server is corrupted whereas the third and the fourth ones required that both the first and the last mix-servers are corrupted.

Chapter 4
Formal Analysis of the WireGuard Protocol

> Proofs are not about absolute truth, they're about reasoning under clearly stated assumptions.
>
> *Shafi Goldwasser*

Contents

Chapter Summary

WireGuard is a Virtual Private Network (VPN) integrated into the Linux 5.6 kernel in March 2020. The protocol inside WireGuard is a dedicated extension of the `IKpsk2` protocol from the Noise Framework. Different analyses of WireGuard and `IKpsk2` protocols have been proposed in the symbolic models with either ProVerif or Tamarin. These analyses, however, consider different adversarial models or refer to simplified versions of the protocols. In this work, we propose a unified formal model of the WireGuard protocol in the symbolic model. Our models are compatible with both the automatic cryptographic protocol verifiers ProVerif and Tamarin using the Sapic$^+$ framework. We consider a full protocol execution which includes the handshake phase, the transport phase and the cookie messages used to protect against denial of service attacks. We model numerous adversarial capabilities, namely adversaries that can read or set static, ephemeral or pre-shared keys, read or set Elliptic Curve Diffie-Hellman (ECDH) pre-computations, and control public key distribution. Eventually, we present our results in a unified and interpretable way, allowing comparisons with previous analyses. Finally, thanks to our models, we give necessary and sufficient conditions for each analyzed security property to be compromised. We confirm a flaw on the anonymity of the communications and point out an implementation choice which considerably weakens the security of the protocol. We then propose remediations that we prove secure.

4.1 Introduction

In 2017, Jason Donenfeld introduced *WireGuard*, an open-source software implementation of Virtual Private Network (VPN) based on the protocol `IKpsk2` from the Noise framework [Don17, Per18]. WireGuard is implemented as a kernel virtual network interface for Linux and was officially integrated into the version 5.6 in 2020 [Lin20]. WireGuard proposes a different approach from other classical VPNs such as IPsec [FK11] or OpenVPN [Ope01] since it does not let users configure cryptographic algorithms and as pointed out by Donenfeld in [Don17]: WireGuard “lacks cipher and protocol agility”. When using WireGuard, the cryptographic algorithms are therefore pre-defined by default.

The protocol, as despicted in Figure 4.1, involves two actors: an *Initiator* and a *Responder*, also referred to as *peers*. WireGuard does not define the notions of a *client* and a *server*. Peers can indifferently play any of the two roles: a peer that starts a session is considered as an Initiator and the other peer as the Responder. The protocol is composed of two phases: a key exchange phase and a transport phase. The first three exchanged messages constitute the *handshake*. A symmetric key is obtained from the first two messages, which it is used to encrypt all subsequent messages. The key exchange phase involves two messages, `InitHello` and `RecHello`, and combines long-term and ephemeral *Diffie-Hellman* values. The transport phase involves one message’s type, `TransData`. WireGuard involves a fourth message type `CookieRep`, for protection against denial of service attacks. The *Handshake* is considered complete *after* the first message from

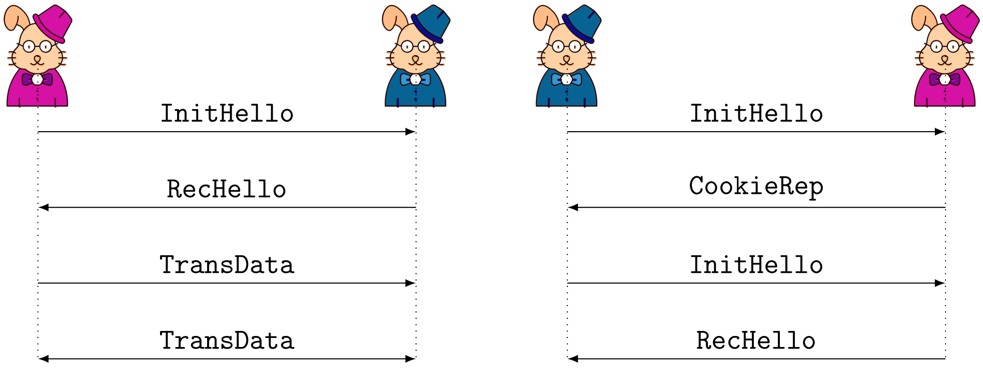


Figure 4.1: The WireGuard protocol messages with cookies (on the right) and without cookies (on the left). The participants of the protocol can play both roles: the Initiator and the Responder.

the transport phase which *must* be sent by the Initiator. Hence, the protocol involves a **1.5-RTT** (Round-Trip Time) key exchange. Also as stated in [Don17], the protocol requires an out of band data share, not considered as part of the protocol. WireGuard is meant to provide peers' mutual authentication, session keys' forward secrecy and "identity hiding "[1].

Different formal analyses of WireGuard were proposed, in both the symbolic and the computational model, and with or without computer-aided proof assistants [DP18, LBB19, DM17]. Even though the analyses point out some weaknesses, they confirm the robust design of the protocol, as most of the claimed security properties are met. These analyses, however, consider different adversarial models or different formulation of the security properties or may refer to simplified versions of the protocol. For instance, the authors of [DM17] conducted a symbolic formal analysis of WireGuard using the Tamarin prover, thus establishing a formal model of the protocol's security properties. Their verification confirmed, for example, that WireGuard satisfies the identity hiding property, demonstrating that the participant identities remain anonymous from adversaries during session establishment. In their symbolic model, the authors abstract WireGuard's message authentication codes (MACs) as fixed and attacker-known strings. This simplification treats the MACs as opaque identifiers rather than cryptographic functions, thus omitting any algebraic properties or symbolic relations related to these functions. While this abstraction deviates from WireGuard's actual use of the keyed Blake2s used for the MACs, it remains a sound abstraction in the symbolic model. In [LBB19], the cryptographic analysis of WireGuard using CryptoVerif points out that the use of static public keys in the MACs in WireGuard negatively affects the identity hiding guarentees. This conclusion could not have been derived using the symbolic modeling of [DM17],

[1] Identity hiding is the term used in Wireguard's whitepaper and website to refer to anonymity of the peers.

as their abstraction lacks the necessary expressiveness to capture static public keys used in the MACs.

Several formal analyses of the `IKpsk2` from the Noise Protocol Framework have been proposed, each considering distinct adversarial capabilities. For instance, Kobeissi et al. [KNB19] analyzed `IKpsk2` under the standard Dolev-Yao attacker model and considering dishonest participants using ProVerif, while Girol et al. [GHS^{+}20] examined the security of the protocol in different scenarios of the attacker's capabilities ranging from standard Dolev-Yao attacker to more enriched scenarios with key compromises using Tamarin.

4.1.1 Our Contributions

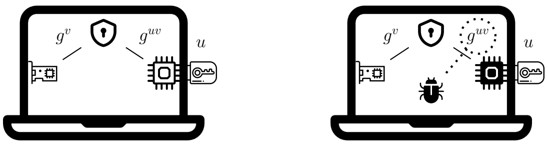
We show that it is possible to model WireGuard more precisely in the symbolic model. Our aim is to propose a new symbolic model that aggregates and enriches all existing models, in terms of the exchanged messages, the adversarial capabilities and the security properties modeling. Our contributions are the following:

- We first review previous analyses of both WireGuard and `IKpsk2`, and we point out disparities between the verified security properties, protocols' models and adversary models. The symbolic analyses involve different models and different automatic protocol verifiers (ProVerif and Tamarin), and computational analyses are manual or use automatic protocol provers (CryptoVerif). This allows us to identify the model from [LBB19] as the most comprehensive model of the WireGuard protocol compared to the protocol's specifications [Don20], and the threat model from [GHS^{+}20] as the model that captures the largest number of adversarial capabilities. We therefore propose a new symbolic model that enriches these analyses with more details of the protocol and more adversary capabilities. We use the framework Sapic^{+}: our model is in the applied Π-Calculus and all the security properties are verified with both ProVerif and Tamarin. Our adversary can read or set static, ephemeral or pre-shared keys, read or set Elliptic Curve Diffie-Hellman (ECDH) pre-computations, and can control public key distribution.

- Inspired by [BC14, GHS^{+}20], we present a dedicated methodology, that is of independent theoretical interest, to assess all adversary models, all key compromise combinations, and synthesize security property results into compact formulas. Each derived formula precisely specifies the exact conditions - relative to the attacker's capabilities - required to violate the target security property. Although many prior works in the literature have expressed the analyzed security properties in a similar manner, they exhibit significant heterogeneity in their formal frameworks, informal methodological approaches, and terminological conventions. Our key contribution is the introduction of a unified methodology and formulation to address this critical gaps in current fragmented analysis practices. This unification enables both direct comparison of the obtained results across different formal analyses for the same protocol, and systematic security evaluation between distinct protocols which facilitates the establishment of the notion of *protocol security hierarchy* presented in [BC14].

- Our models allow us to confirm a flaw related to the anonymity of the peers. This flaw has previously been identified in a computational analysis of the protocol [LBB19], yet not captured in the symbolic analysis of [DM17] due to the high level of abstraction of the MAC used in the models. We show that it is possible to find this attack against anonymity in the symbolic model considering a more precise protocol model. This flaw allows an attacker to identify a VPN user even if this user hides behind an access point, because this flaw is related to the protocol design and does not rely on network mechanisms. We propose fixes to this flaw which we prove secure in the symbolic model.

- Following the source code of WireGuard, we choose to add a new adversarial capability in our threat model which is not captured by previous models. In fact, to speed-up computations, each peer computes the ECDH product between its private key and the other peers' public keys at the interface setup once, and keeps these products in memory while the interface is up. Peers have a specific field that contains pre-computed ECDH products. In the Linux Kernel, for example, this field is named `precomputed_static_static` and named `precomputedStaticStatic` in the user-land Go implementation [Don23]. At the interface initialization, all peers public keys are read and the ECDH product between the interface private key and the public key is computed, and the field's value is set with the result of the computation. Thanks to our model, we find out that this implementation optimization allows new attack scenarios if the adversary can get access to the ECDH pre-computation.



(a) Uncompromised memory.

(b) Compromised memory.

Figure 4.2: Potential Vulnerability Against Wireguard.

To illustrate the importance of the pre-computation assessment, we describe in Figure 4.2 a potential vulnerability against one WireGuard implementation. The default storage for static private keys is in the configuration file. Hence, an attacker that accesses these files compromises static private keys. To mitigate this risk, static private keys can be generated and stored in a smart card which can perform ECDH computations. An example of such card is OpenPGP embedded in YubiKey [Lud23]. The security purpose in this context is to protect against an attacker that has access to files and memory of the WireGuard process, but shall not be able to compromise static keys stored in the smart card. In this implementation, OpenPGP on YubiKey generates and stores a static private key, requiring ECDH support, which is available on recent YubiKey models. Then Go implementation of WireGuard is used, which provides a full user-space implementation. Such an architecture aims at mitigating memory leakage, as the static key is protected by the smart card. Once the interface is mounted, an attacker could however access the ECDH product in process memory, due to the pre-computation. Precisely, we consider an Initiator of a static private key u, embedded in a smart card 🔑, which uses a WireGuard client 🛡 and a network configuration 🖧 that contains the Responder's public key g^v. When the smart card is plugged and the WireGuard interface is mounted, the ECDH product g^{uv} is pre-computed in memory ▣. In a safe environment, depicted in Figure 4.2a, this pre-computation is not compromised, however, in an unsafe environment, depicted in Figure 4.2b, an attacker 🐞 can corrupt the memory ▣ and the pre-computed ECDH product g^{uv}, while the private key u in the smart card 🔑 remains safe. Our contribution is to consider a symbolic adversary model that enhances the adversarial capabilities from [GHS$^+$20] with pre-computations access and modification. Note that the compromise of the ECDH pre-computation is *weaker* than the compromise of the private static keys: if an adversary has access the private key, the adversary knows the ECDH product, however, the opposite is false. We show that in contradiction with this, an adversary that has access to the pre-computation is *as powerful* as an adversary that has access to static private keys in many attack scenarios against WireGuard.

4.1.2 Related Work

The first - and, prior to our work, the only - symbolic formal analysis of WireGuard was conducted in [DM18] using Tamarin. In this model, all the security properties intended to be guaranteed by WireGuard were formally verified. Beyond relying on the assumption that public keys are unknown to the attacker, the model makes many significant abstractions, e.g., it completely idealizes the MACs, omits cookie-handling mechanisms, and adopts a weak definition of anonymity, i.e., anonymity is formalized and analyzed as a trace property. In the computational model, WireGuard was analyzed with CryptoVerif in [LBB19] and "manually "in [DP18]. WireGuard uses a dedicated protocol that relies on an ECDH key exchange from the Noise framework [Per18], named `IKpsk2`. Protocols from this framework have also been analyzed in the symbolic model: the analyses have been proposed with ProVerif [KNB19] and with Tamarin [SD18], which allowed to confirm claimed security properties from the framework. A complementary analysis is proposed in [GHS^{+}20], also with Tamarin. We give further details about each model in Section 4.3. In line with the (non computer-aided) analysis done on WireGuard, an analysis of the Noise protocols in the computational model is presented in [DRS20].

Security Properties and Threat Scenarios. While researchers in symbolic protocol verification have largely converged on common definitions and formalizations of security properties, no such agreement exists about how the results of the analysis should be presented nor described. The most commonly employed approach is to ascertain whether a protocol meets a security property under predetermined assumptions regarding the attacker's capabilities, as demonstrated by the overwhelming majority of analyses, e.g., [KNB18, BCW22, CCD23a, CMR23] and [GGCG^{+}21]. A more comprehensive analysis entails conducting a deeper verification to determine what an attacker can do to compromise the security property or what is the most potent threat model that the protocol can still withstand, instead of only checking if the property is verified or not. In [BC14], the authors presented the concept of a protocol security hierarchy by decomposing each security property into a *basic* one, e.g., secrecy and authentication, and the set of the adversarial capabilities that the protocol is resilient to with regard to the basic security property. The authors of [GHS^{+}20], based on the work of [BC14], introduce *the strongest threat model* under which each security property is verified, thereby, ensuring that the analyzed protocols are secure in the presence of weaker threat models. In [JKKR23], the authors identified both maximal threat scenarios for the security properties to hold and minimal threat scenarios that violate the property. In [BJKS24], Bhargavan et al. provided the so-called *maximal formulation of the security properties* which ties in with the idea of [GHS^{+}20]. Since the described methodology of [GHS^{+}20] is bind to the analyzed protocols from the Noise framework, and the methodologies followed by [JKKR23] and [BJKS24] are informal, it is not clear whether the notions of maximality and the minimality described in all subsequent works are the same. The absence of shared frameworks, methodological alignment, and terminological consistency across these works hardens the comparison between them. In this work, we propose a unified formalization for expressing the security properties as compact formulas considering adversarial compromises.

Outline. We begin by giving a detailed description of WireGuard in Section 4.2. In Section 4.3, we assess previous analyses of both WireGuard and `IKpsk2`, and point out disparities between the models. In Section 4.4, we present our methodology. Then, in Section 4.5, we describe our new model, the assesed security properties, and we present the results of our analyses. Finally, in Section 4.6, we conclude and outline directions for future work.

4.2 The WireGuard Protocol

In the following, we give a full description of the WireGuard protocol. We begin by enumerating the cryptographic primitives used within the protocol. Then, we describe the exchanged messages contents and how they are computed by each participant of the protocol.

Cryptographic Primitives. WireGuard uses a cyclic group $\mathbb{G}$, of generator g and a closed set of cryptographic primitives: a hash function Hash, two message authentication codes HMAC and MAC, three key derivations $\mathsf{KDF}_1, \mathsf{KDF}_2, \mathsf{KDF}_3$, two authenticated encryption algorithms AEAD and XAEAD, and a padding scheme pad. The Initiator and the Responder use the following material: $(u, U = g^u)$ is the Initiator static key pair, $(x, X = g^x)$ is the Initiator ephemeral key pair, $(v, V = g^v)$ is the Responder static key pair, $(y, Y = g^y)$ is the Responder ephemeral key pair, ts is a timestamp, psk is an optional pre-shared symmetric key, and C, I and M are public constants. In WireGuard, these are instantiated as follows:

- $\mathbb{G}$ is the group of points of the elliptic curve $\mathsf{Curve25519}$ [Ber06], [LHT16].
- $h \leftarrow \mathsf{Hash}(I)$ is the computation of a 32-byte fingerprint h from an input I with the hash function $\mathsf{Blake2s}$ [ANWW13].
- $M \leftarrow \mathsf{HMAC}(K, I)$ is the computation of a 32-byte message authentication code M from a key K and an input I with the hash function $\mathsf{Blake2s}$ as described in [HMA02].
- $M \leftarrow \mathsf{MAC}(K, I)$ is the computation of a 16-byte message authentication code M from an input I and a key K with $\mathsf{Blake2s}$ hash function, as described in [SA15].
- $\tau_1 \leftarrow \mathsf{KDF}_1(K, I)$, $(\tau_1, \tau_2) \leftarrow \mathsf{KDF}_2(K, I)$ and $(\tau_1, \tau_2, \tau_3) \leftarrow \mathsf{KDF}_3(K, I)$ are key derivations from [Kra10], where K is a key and I an input.
- $(C, T) \leftarrow \mathsf{AEAD}(K, N, P, A)$ is the authenticated encryption algorithm from [LN15], which combines $\mathsf{ChaCha20}$ and $\mathsf{Poly1305}$ algorithms. From a key K, a 12-byte nonce N, a plaintext P, and authentication data A, it

computes a ciphertext C of length $|P|$ bytes and a 16-bytes authentication tag T, hence its total byte-length is $|P| + 16$. The 12-bytes nonce N is a concatenation of 0s followed by an 8-byte counter.

- $(C, T) \leftarrow \mathsf{XAEAD}(K, N, P, A)$ is an authenticated encryption algorithm which is a variant of the previous one where the nonce N is a random 24-bytes string.
- $P \| 0 \ldots 0 \leftarrow \mathsf{pad}(P)$ is the padding algorithm for an input P.

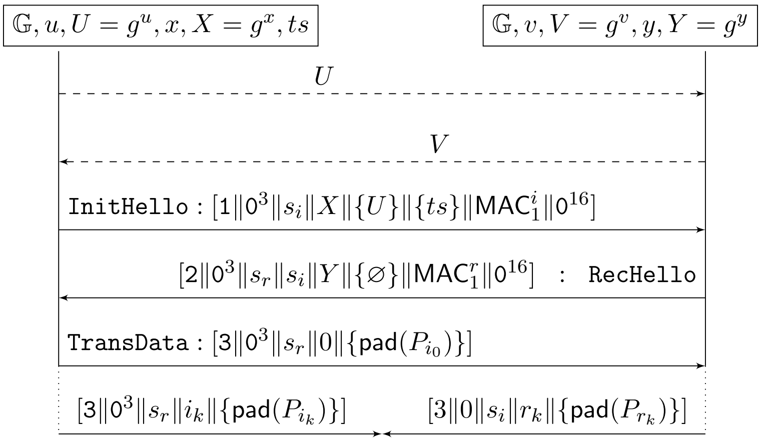

$\mathbb{G}, u, U = g^u, x, X = g^x, ts$
$\mathbb{G}, v, V = g^v, y, Y = g^y$
U
V
InitHello : $[1\|0^3\|s_i\|X\|\{U\}\|\{ts\}\|\mathsf{MAC}_1^i\|0^{16}]$
$[2\|0^3\|s_r\|s_i\|Y\|\{\varnothing\}\|\mathsf{MAC}_1^r\|0^{16}]$: RecHello
TransData : $[3\|0^3\|s_r\|0\|\{\mathsf{pad}(P_{i_0})\}]$
$[3\|0^3\|s_r\|i_k\|\{\mathsf{pad}(P_{i_k})\}]$
$[3\|0\|s_i\|r_k\|\{\mathsf{pad}(P_{r_k})\}]$

Figure 4.3: WireGuard (without cookies).

Message content and computation. The messages of the WireGuard protocol are depicted in Figure 4.3 and Figure 4.4. In the `InitHello` message 1 and 0 are constant bitstrings, 0^3 is the concatenation of 3 0s, s_i is a random session identifier, X is the Initiator's ephemeral key, U is the Initiator's static public key, ts is a timestamp, and MAC_1^i is a first message authentication code. Depending on `CookieRep` message, MAC_2^i takes two values: either 0^{16} (the concatenation of 16 0s) or a second message authentication code. To compute the `InitHello` message, the Initiator uses two public values C and I, generates a random session identifier s_i, computes successive hash values h_t, key values k_t and chaining values c_t as follows:

- $ck = \mathsf{Hash}(\mathsf{C})$
- $h_0 = \mathsf{Hash}(ck\|\mathsf{I})$
- $h_1 = \mathsf{Hash}(h_0\|V)$
- $c_0 = \mathsf{KDF}_1(ck, X)$
- $h_2 = \mathsf{Hash}(h_1\|X)$
- $(c_1, k_1) = \mathsf{KDF}_2(c_0, g^{xv})$
- $\{U\} = \mathsf{AEAD}(k_1, 0, h_2, U)$
- $h_3 = \mathsf{Hash}(h_2\|\{U\})$

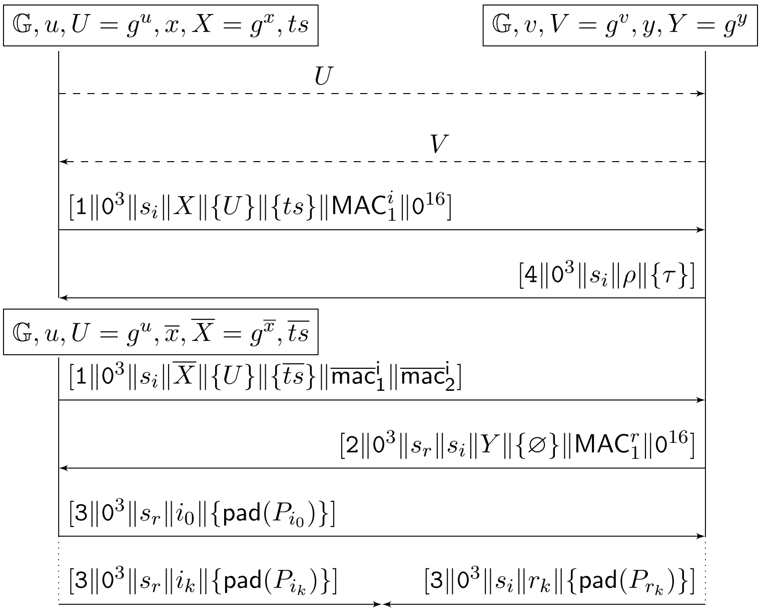

$\mathbb{G}, u, U = g^u, x, X = g^x, ts$
$\mathbb{G}, v, V = g^v, y, Y = g^y$
U
V
$[1\|0^3\|s_i\|X\|\{U\}\|\{ts\}\|\mathsf{MAC}_1^i\|0^{16}]$
$[4\|0^3\|s_i\|\rho\|\{\tau\}]$
$\mathbb{G}, u, U = g^u, \overline{x}, \overline{X} = g^{\overline{x}}, \overline{ts}$
$[1\|0^3\|s_i\|\overline{X}\|\{U\}\|\{\overline{ts}\}\|\overline{\mathsf{mac}}_1^i\|\overline{\mathsf{mac}}_2^i]$
$[2\|0^3\|s_r\|s_i\|Y\|\{\varnothing\}\|\mathsf{MAC}_1^r\|0^{16}]$
$[3\|0^3\|s_r\|i_0\|\{\mathsf{pad}(P_{i_0})\}]$
$[3\|0^3\|s_r\|i_k\|\{\mathsf{pad}(P_{i_k})\}]$
$[3\|0^3\|s_i\|r_k\|\{\mathsf{pad}(P_{r_k})\}]$

Figure 4.4: WireGuard (with cookies).

- $(c_2, k_2) = \mathsf{KDF}_2(c_1, g^{uv})$
- $\{ts\} = \mathsf{AEAD}(k_2, 0, h_3, ts)$
- $h_4 = \mathsf{Hash}(h_3\|\{ts\})$

Finally, a message authentication code is appended to the message, computed on the bitstring $[1\|0^3\|s_i\|X\|\{U\}\|\{ts\}]$, with key $\mathsf{Hash}(\mathsf{M}\|V)$ (i.e., key is derived from public value M and the Responder public key V), and the `InitHello` message. At reception, the Responder performs the necessary computations to obtain the same hash and key values, decrypts $\{U\}$, checks that U is legitimate and decrypts the encrypted timestamp $\{ts\}$.

In the message `RecHello`, $\mathsf{2}$ is a constant bitstring, s_r, s_i are session identifiers, Y is the Responder's ephemeral key, $\{\varnothing\}$ is the encryption of the empty string, and MAC_1^r and MAC_2^r are similar as for `InitHello` message. In the `TransData` messages $\mathsf{3}$ is a constant bitstrings, s_i and s_r are session identifiers, $\{\mathsf{pad}(P_{i_k})\}$ and $\{\mathsf{pad}(P_{r_k})\}$ are the padded and encrypted payloads. Finally, in the message `CookieRep`, $\mathsf{4}$ is a constant bitstrings, s_i is a session identifier, ρ is a random nonce and $\{\tau\}$ is the encrypted cookie. To compute `RecHello`, the Responder generates a random session identifier s_r, computes the next hash values h_t, key values k_t and chaining values c_t as follows:

- $c_3 = \mathsf{KDF}_1(c_2, Y)$
- $h_5 = \mathsf{Hash}(h_4\|Y)$
- $c_4 = \mathsf{KDF}_1(c_3, g^{xy})$
- $c_5 = \mathsf{KDF1}(c_4, g^{uy})$

- $(c_6, h_{rt}, k_6) = \mathsf{KDF}_3(c_5, 0)$
- if $\mathsf{psk} = \varnothing$, $(c_6, h_{rt}, k_6) = \mathsf{KDF}_3(c_5, \mathsf{psk})$ if $\mathsf{psk} \neq \varnothing$, $h_6 = \mathsf{Hash}(h_5 \| h_{rt})$
- $\{\varnothing\} = \mathsf{AEAD}(k_6, 0, h_6, \varnothing)$
- $h_7 = \mathsf{Hash}(h_6 \| \{\varnothing\})$

Similarly as for the `InitHello`, a message authentication code is appended to the message, computed on the bitstring $[\mathsf{2} \| \mathsf{0}^3 \| s_r \| s_i \| Y \| \{\varnothing\}]$, with key $\mathsf{Hash}(\mathsf{M}, U)$ (i.e., key is derived from public value M and the Initiator public key U). At reception, the Initiator performs the necessary computations to obtain the same hash and key values, decrypts $\{\varnothing\}$ and checks that the obtained value is $\varnothing$.

After the `InitHello` and `RecHello`, both the Initiator and the Responder share a common session key k_6. From this key they derive two keys $(C^i, C^r) = \mathsf{KDF}_2(k_6, \varnothing)$ and use these keys, respectively, to protect data from the Initiator to the Responder and from the Responder to the Initiator.

To compute the first `TransData` message, which is from the Initiator to the Responder, the Initiator takes the received the session identifier s_r, the current counter value i_k, and computes $\{\mathsf{pad}(P_{i_k})\} = \mathsf{AEAD}(C^i, i_k, \mathsf{pad}(P_{i_k}), \varnothing)$ where P_{i_k} is the plaintext sent at this step and $\mathsf{pad}(P_{i_k})$ is the padded plaintext. The Responder performs same computation with the Initiator session identifier s_i, the current counter value r_k, and the plaintext P_{r_k}. Note that the first transport message is always from the Initiator to the Responder. For WireGuard, the counter maximal value is 2^{60} (i.e., at most 2^{60} transport messages are encrypted with same session key).

The WireGuard protocol embeds a protection against denial of service, based on the `CookieRep` messages. To build such a message, WireGuard uses information from transport layer, as messages are transported in UDP datagrams [Pos80]. The `InitHello` message is transported in the following packet $[\mathtt{IP}_i \| \mathtt{IP}_r \| \mathtt{Port}_i \| \mathtt{Port}_r \| \mathtt{InitHello}]$ where $\mathtt{IP}_i$ and $\mathtt{Port}_i$ are the public IP and port for the Initiator, and $\mathtt{IP}_r$ and $\mathtt{Port}_r$ are the public IP and port for the Responder. The Responder generates a random value R_m, uses $\mathtt{IP}_i$ and $\mathtt{Port}_i$ from the incoming packet and computes the cookie value $\tau = \mathsf{MAC}(R_m, \mathtt{IP}_i \| \mathtt{Port}_i)$. This cookie is then encrypted: the Responder generates a random nonce ρ and computes $\{\tau\} = \mathsf{XAEAD}(\mathsf{Hash}(V), \mathsf{MAC}_1^i, \rho, \tau)$, where MAC_1^i is extracted from the `InitHello` message. At reception, the Initiator decrypts τ, generates a new `InitHello` message, with the same session identifier s_i, a new ephemeral key pair $\overline{x}, \overline{X} = g^{\overline{x}}$, a new timestamp $\overline{ts}$ and a new authentication code $\overline{\mathsf{mac}_1^{\mathsf{i}}}$ as before, except that now it appends a second authentication code to the message, computed on the first 7^{th} fields $[\mathtt{1} \| \mathsf{0}^3 \| s_i \| \overline{X} \| \{U\} \| \{\overline{ts}\} \| \overline{\mathsf{mac}_1^{\mathsf{i}}}]$, with key the cookie value τ. At reception, the Responder verifies this additional authentication code and continues the protocol as before.

WireGuard uses `IKpsk2` (and not `IK`, nor `KK`, nor `KKpsk2`). The Noise framework [Per18] defines a set of key exchange protocols, among which are the protocols `IK`, `KK`, `IKpsk2`, and `KKpsk2`. These four protocols are referenced in the WireGuard documentation and source code as the basis for the key exchange protocol inside WireGuard. At first glance, as pointed in [AMW19], it seems that WireGuard is closer to `KKpsk2` than `IKpsk2` because of the initial out of band public keys exchange. However, in `KKpsk2`, the Initiator knows to whom it sends `InitHello` message and Responder *knows from whom it receives it*, whereas in `IKpsk2`, the Initiator knows to who it sends `InitHello` message but Responder does not know from whom it receives it. An application built upon `KKpsk2` shall ensure both parties know to whom they communicate before starting key exchange, whereas an application built upon `IKpsk2` can accept the Responder does not know *a priori* who sends an `InitHello` message. Yet, the Responder shall be able to assess if the received message is acceptable. This is exactly the path followed by WireGuard: each party has a set of acceptable peers (a list of acceptable public keys), but *discovers* who initiates a key exchange and *checks* if it is acceptable during key exchange. WireGuard lets peers use an optional pre-shared key that shall be shared beforehand. When this option is not chosen (which is described as $\mathsf{psk} = \varnothing$), WireGuard still implements `IKpsk2`: a difference between `IK` and `IKpsk2` is that for `IKpsk2` ephemeral keys X and Y are included in the derivation of the session key, which is not the case for `IK`. As a consequence, the protocol from the Noise framework we compare to WireGuard is `IKpsk2`. We need however to point out similarities and differences.

Similarities between WireGuard and `IKpsk2`. The definition of `IKpsk2` uses the same set of abstract algorithms as WireGuard (a cyclic group $\mathbb{G}$, of generator g, a hash function Hash, message authentication codes HMAC and MAC, key derivations $\mathsf{KDF}_1, \mathsf{KDF}_2, \mathsf{KDF}_3$, authenticated encryption algorithms AEAD and a padding scheme pad) except that `IKpsk2` does not instantiate them. It is up to the application based on `IKpsk2` to choose cryptographic primitives, as WireGuard does. The computation of keys for `IKpsk2` is similar to WireGuard. `IKpsk2` relies on an initial out of band pre-message from the Responder to the Initiator in which the Responder sends its static public key and conversely the Initiator sends its static public key in first message. `IKpsk2` does not specify how these keys are validated: indeed, the Noise specification states *it's up to the application to determine whether the remote party's static public key is acceptable.* The WireGuard specification is similar as it states *WireGuard rests upon peers exchanging static public keys with each other.*

Differences between WireGuard and `IKpsk2`. WireGuard involves a **1.5-RTT** key exchange: after messages `InitHello` and `RecHello` are correctly sent and received, a first `TransData` shall be sent by the Initiator. After this first `TransData` message any peer can send other `TransData` messages. This feature is however not mandatory in Noise protocols and hence in `IKpsk2`: in `IKpsk2`, after the handshake, transport messages can be sent both from the Initiator to the Responder or from the Responder to the Initiator. This feature is captured differently in previous analyses.

In `IKpsk2`, the first message is $[X \| \{U\} \| \{m_0\}]$, whereas the `InitHello` message in WireGuard message is $[\mathtt{1} \| \mathsf{0}^3 \| s_i \| X \| \{U\} \| \{ts\} \, \| \mathsf{MAC}_1^i \| \mathsf{MAC}_2^i]$, where MAC_2^i can equal $\mathsf{0}^{16}$. Hence with $m_0 = ts$, the messages are similar, however they differ: `InitHello` has the header $[\mathtt{1} \| \mathsf{0}^3]$, embeds a session identifier s_i, and the fields MAC_1^i and MAC_2^i. Similarly, in `IKpsk2` the second message is $[Y \| \{m_1\}]$, which is different from the WireGuard `RecHello`. Hence with $m_1 = \varnothing$, messages are similar, but differ due to the header, session identifiers and MAC_1^r and MAC_2^r fields. Finally, transport messages also differ as WireGuard `TransData` includes a header, a session identifier and transmits the counter in clear. Note that the Noise specification [Per18] allows this clear counter transmission.

4.3 Landscape of Prior Analysis of WireGuard

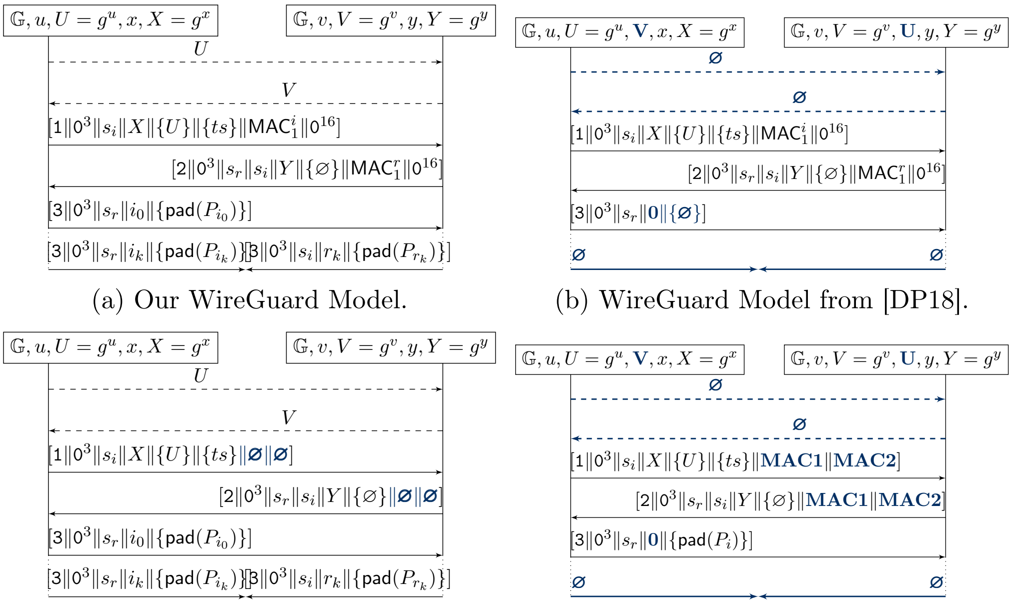
In this section, we provide a comprehensive overview of the security properties examined in prior analyses, along with the formal models used to represent the analyzed protocols. We also detail the adversarial models considered in these works, which define the capabilities and limitations of potential attackers. By synthesizing these aspects, we identify the most precise protocol model - the one that captures the highest level of details - alongside the most prevalent threat model, which encompasses the widest range of compromise scenarios. These selections form the basis for our modeling, which seeks to refine and extend existing models by incorporating more protocol specifics and more exhaustive adversarial scenarios, thereby strengthening the robustness of security analysis.

4.3.1 Protocol Models used Previous Analyses

We point disparities between previous protocol models of WireGuard. The different models did not account for all elements in all messages, as illustrated in Figure 4.5. In Figure 4.5a, we describe our model which aims at including all the protocol's specifics and details included in previous models. We note that on the left side of Figure 4.5, all models include an initial key distribution (which can be potentially compromised), while on the right side, all models assume a safe out-of-band initial key distribution.

Figure 4.5b which refers to the model of [DP18], models a protocol composed of three messages: two first messages `InitHello`, `RecHello`, and a transport message from the Initiator to the Responder that does not correspond exactly to WireGuard as the encrypted data is ∅. Although this model of the protocol was used to perform a computational analysis, we describe it here because we are only interested in the protocol steps modeled. Figure 4.5c depictes the protocol model from [LBB19], it includes the pre-messages for static key (U and V) distribution. The two first messages correspond to `InitHello` and `RecHello` messages, but without the corresponding message authentication codes. The first transport message that is in one direction, and then the other transport messages `TransData` that can be in both directions, with the counter in clear. This model was also used for a computational analysis of the protocol. In Figure 4.5d, the model from the symbolic analysis of [DM18] is depicted. It is composed of three messages: two first messages that do not correspond exactly to WireGuard as the message authentication codes in both `InitHello` and `RecHello` are replaced by constant bitstrings **MAC1** and **MAC2**, followed by the first transport message `TransData` from the Initiator to the Responder.

It appears that the most precise model of the protocol, namely the model that encompasses more details about the protocol, is one from [LBB19] which is however still incomplete.


$\mathbb{G}, u, U = g^u, x, X = g^x$
$\mathbb{G}, v, V = g^v, y, Y = g^y$
U
V
$[1\|0^3\|s_i\|X\|\{U\}\|\{ts\}\|\mathsf{MAC}_1^i\|0^{16}]$
$[2\|0^3\|s_r\|s_i\|Y\|\{\varnothing\}\|\mathsf{MAC}_1^r\|0^{16}]$
$[3\|0^3\|s_r\|i_0\|\{\mathsf{pad}(P_{i_0})\}]$
$[3\|0^3\|s_r\|i_k\|\{\mathsf{pad}(P_{i_k})\}]$
$[3\|0^3\|s_i\|r_k\|\{\mathsf{pad}(P_{r_k})\}]$
$\mathbb{G}, u, U = g^u, \mathbf{V}, x, X = g^x$
$\mathbb{G}, v, V = g^v, \mathbf{U}, y, Y = g^y$
$\varnothing$
$\varnothing$
$[1\|0^3\|s_i\|X\|\{U\}\|\{ts\}\|\mathsf{MAC}_1^i\|0^{16}]$
$[2\|0^3\|s_r\|s_i\|Y\|\{\varnothing\}\|\mathsf{MAC}_1^r\|0^{16}]$
$[3\|0^3\|s_r\|\mathbf{0}\|\{\varnothing\}]$
$\varnothing$
$\varnothing$
$\mathbb{G}, u, U = g^u, x, X = g^x$
$\mathbb{G}, v, V = g^v, y, Y = g^y$
U
V
$[1\|0^3\|s_i\|X\|\{U\}\|\{ts\}\|\varnothing\|\varnothing]$
$[2\|0^3\|s_r\|s_i\|Y\|\{\varnothing\}\|\varnothing\|\varnothing]$
$[3\|0^3\|s_r\|i_0\|\{\mathsf{pad}(P_{i_0})\}]$
$[3\|0^3\|s_r\|i_k\|\{\mathsf{pad}(P_{i_k})\}]$
$[3\|0^3\|s_i\|r_k\|\{\mathsf{pad}(P_{r_k})\}]$
$\mathbb{G}, u, U = g^u, \mathbf{V}, x, X = g^x$
$\mathbb{G}, v, V = g^v, \mathbf{U}, y, Y = g^y$
$\varnothing$
$\varnothing$
$[1\|0^3\|s_i\|X\|\{U\}\|\{ts\}\|\mathbf{MAC1}\|\mathbf{MAC2}]$
$[2\|0^3\|s_r\|s_i\|Y\|\{\varnothing\}\|\mathbf{MAC1}\|\mathbf{MAC2}]$
$[3\|0^3\|s_r\|\mathbf{0}\|\{\mathsf{pad}(P_i)\}]$
$\varnothing$
$\varnothing$

(a) Our WireGuard Model.

(b) WireGuard Model from [DP18].

(c) WireGuard Model from [LBB19].

(d) WireGuard Model from [DM18].

Figure 4.5: Comparison with other models, where for each model, blue bold denotes part of the protocol that is precisely defined in our model but not in the model, hence for each model, differences with our model are highlighted.

		Reference			
		[DM18]	[DP18]	[LBB19]	This work
Method	Pen-and-Paper	✗	✓	✗	✗
	CryptoVerif	✗	✗	✓	✗
	Tamarin	✓	✗	✗	✓
	ProVerif	✗	✗	✗	✓
Adversary Model	Static private key access	✓	✓	✓	✓
	Static private key modification	✗	✗	✓	✓
	Ephemeral private key access	✓	✓	✓	✓
	Ephemeral key modification	✗	✗	✗	✓
	Pre-shared key access	✓	✓	✓	✓
	Pre-shared key modification	✗	✗	✓	✓
	Key distribution compromise	✗	✗	✓	✓
	Pre-computation access	✗	✗	✗	✓
	Pre-computation modification	✗	✗	✓	✓

✓: present ✗: not considered or absent

Table 4.1: Adversary Models and Proofs Techniques.

4.3.2 Adversary Models

We also point out disparities between *adversary models* considered in previous analysis of the protocol. The details about each considered adversary model in previous analysis of WireGuard are given in Table 4.1. We also compare adversary models used in the analysis of `IKpsk2`. In summary, all the works from [DM18], [KNB19] and [DP18] capture the security against key leakage and only consider the scenario where keys are generated honestly, wherease the models from [GHS$^+$20] and [LBB19] capture both key leakage *and* key modification.

Finally [KNB19] captures static and pre-shared key compromise while all others [DM18, GHS$^+$20, DP18, LBB19], capture static, ephemeral and pre-shared keys compromise. The adversary model from [GHS$^+$20], adapted to `IKpsk2`, is the most prevalent model as it captures key corruption through leakage and modification. As stated previously, we enrich this adversary model with new adversarial capabilities related to leakage of pre-computation.

4.3.3 Security Properties Defined in Previous Analyses

Symbolic analyses of WireGuard are only proposed in [DM18] with the Tamarin prover. The authors analyzed *Correctness*, *Key Agreement*, *Key Secrecy*, *Session Uniqueness*, *Identity Hiding* and *weak Forward Secrecy* which were defined as trace properties. The difference between *weak Forward Secrecy* (from [DM18]) and *Forward Secrecy* is that the former is defined for a passive adversary while the latter is analyzed for an active adversary. *Key Secrecy* means that the session key is not known to the adversary. *Session Uniqueness* means that different sessions have different keys. Each security property is tested against a unique key compromise scenario, for which the test succeeds, however this has the strong limitation that other key compromise scenarios are not included in the analyses. Our contribution is to provide the analysis with regard to *a large set of* key compromise scenarios.

Symbolic analyses of `IKpsk2` are proposed in [KNB19] (with ProVerif) and [GHS$^+$20] (with Tamarin). [KNB19] defines agreement and secrecy to fit the properties that are informally described in [Per18], as trace properties. This leads to a restricted analysis as the resulting definitions are only tested against a specific key compromise scenario, hence this analysis shares the same limitation as [DM18] and in [HNS$^+$21]. As opposed to all previous analysis, [GHS$^+$20] proposes a different approach: analyze protocols from the Noise framework against security properties which are *not* the ones informally defined in [Per18], but are precise standard properties: *secrecy of payloads*, *non-injective agreement* and *injective agreement* on messages as defined in [Low97], and *anonymity.* Secrecy and agreement are modeled as trace properties while anonymity is modeled as an equivalence property. Furthermore, this analysis assesses *a large set of* key compromise scenarios, including a fine-grained assessment of forward secrecy, which depends on both static keys but also on pre-shared key. We use this analysis as a reference for our adversarial model in Sapic$^+$ and we enhance it to include the leakage of the pre-computed ECDH keys.

4.4 Offensive, Defensive Models

In the subsequent section, we adopt the standard conventions of set theory and first-order logic. Any modifications to the conventional meaning of symbols will be explicitly stated. New notation will be systematically introduced and defined as needed.

Remark:

When using sets, we adhere to the standard definition where sets contain only distinct elements. Thus, any enumeration with duplicate elements (e.g., $\{a, b, b\}$) is equivalent to its reduced form ($\{a, b\}$).

When defining an *atomic capability* of an adversary, it is implicitly assumed that the adversary is also endowed with all the power of a Dolev-Yao [DY83] attacker.

Definition 4.4.1 (Atomic Capability)**.** *Let $\mathcal{P}$ be a security protocol, $\mathcal{D}_{\mathcal{P}}$ be a finite set of data related to $\mathcal{P}$, and $\mathcal{A}$ an adversary. An* atomic capability *of $\mathcal{A}$ is defined for every $d \in \mathcal{D}_{\mathcal{P}}$ as follows:*

- *$\emptyset$ when the attacker has no atomic capability,*
- *R_d when d is generated or computed honestly, yet revealed to the adversary $\mathcal{A}$ at some time during the execution of the protocol $\mathcal{P}$,*
- *R_d^* when d is generated or computed honestly, yet revealed to the adversary $\mathcal{A}$ after the execution of the protocol $\mathcal{P}$,*
- *M_d when d is generated or computed dishonestly, or modified by the adversary $\mathcal{A}$ at some time during the execution of the protocol $\mathcal{P}$.*

Remark:

Our definition of atomic capabilities is based on a constrained list of labels as these represent the only capabilities incorporated in our current analysis framework. However, this set can be easily extended to incorporate additional labels with distinct semantic interpretations as needed.

In Definition 4.4.1, data can be any ground term related to the protocol $\mathcal{P}$ and not only atomic names such as keys or nonces. The intuition behind terms, is that an attacker can compromise for example a session key computed from the application of function symbols on names and constants, without compromising private names themselves.

Let $\mathcal{S} = \{s_1, \ldots, s_k\}$ be a subset of Γ. We denote by $\mathcal{S}^{\wedge}$ and $\mathcal{S}^{\vee}$ the conjunction and respectively the disjunction of all elements in $\mathcal{S}$, defined as:

$$\mathcal{S}^{\wedge} := \bigwedge_{i=1}^{k} s_i \quad \text{and} \quad \mathcal{S}^{\vee} := \bigvee_{i=1}^{k} s_i$$

To enable a systematic security analysis of protocols, we first require a formal adversary model specification.

Definition 4.4.2 (Adversary Model)**.** *Let $\mathcal{P}$ be a protocol and Γ be the set of all atomic capabilities. An adversary model is any subset $\mathcal{S}$ of Γ.*

For example, the set $\{R_x, R_z, M_y\}$ defines an adversary model where the Dolev-Yao attacker is endowed with the atomic capabilities R_x, R_z and M_y.

Given a protocol $\mathcal{P}$, an adversary model $\mathcal{A}$, and a security property φ, abusing notation, we write $\mathcal{P}_{\mathcal{A}} \vDash \varphi$ if the security property φ is satisfied for the protocol $\mathcal{P}$ considering the adversary model $\mathcal{A}$. Since distinct security properties may need different adversarial capabilities to be violated, and because not every adversary model is relevant to all security properties, we introduce the following targeted adversary model specification:

Definition 4.4.3 (Offensive Adversary Model)**.** *Let $\mathcal{S}$ be a subset of Γ. $\mathcal{S}$ is an* offensive model *for the security property φ within the protocol $\mathcal{P}$ with regard to the set of the atomic capabilities Γ, which we note $\overset{\circ}{\mathcal{S}}_{\mathcal{P},\Gamma,\varphi}$, if we have that $\mathcal{P}_{\mathcal{S}} \nvDash \varphi$.*

In presence of an offensive adversary model, the property φ is not satisfied. If we have $\overset{\circ}{\mathcal{S}}_{\mathcal{P},\Gamma,\varphi}$ and $\mathcal{S} \subseteq \mathcal{S}'$ then $\overset{\circ}{\mathcal{S}'}_{\mathcal{P},\Gamma,\varphi}$, namely if a security property is violated by an adversary model, then it is violated by all stronger adversary models (where strength is ordered by capability inclusion). To enable rigorous security analysis, we require a canonical representation of adversary models.

Characterizing security properties only through offensive adversary models is insufficient for comprehensive analysis as one can find the offensive model $\mathcal{A}$ for a security property φ, and another can find $\mathcal{A}' = \mathcal{A} \cup \{a\}$ such as a is an atomic capability not in $\mathcal{A}$.

Definition 4.4.4 (Minimal Offensive Adversary Model)**.** *$\overset{\circ}{\mathcal{S}}_{\mathcal{P},\varphi}$ is said to be* minimal*, and refered to as $\overset{\bullet}{\mathcal{S}}_{\mathcal{P},\Gamma\varphi}$, if it satisfies the following conditions:*

- *$\mathcal{S} = \emptyset$, or*
- *$\mathcal{S} \neq \emptyset$ and $\forall s \in \mathcal{S}$, the set $\mathcal{S} \setminus \{s\}$ is not an offensive adversary model for φ within the protocol $\mathcal{P}$ with regard to the set of atomic capabilities Γ.*

Remark:

If a property holds regardless of the adversary capabilities, then there exists no offensive adversary, and in particular no minimal offensive model. Thus, an empty minimal offensive adversary model corresponds to the Dolev-Yao attacker.

These models represent the smallest possible sets of adversarial capabilities, with respect to set inclusion, that are sufficient to violate a given security property. A model is deemed minimal if it satisfies two key conditions: it must contain enough capabilities to break the security property, yet no proper subset of those capabilities should remain sufficient to do so. This approach provides a systematic framework for identifying fundamental vulnerabilities, as it isolates the exact adversarial atomic capabilities required to compromise the security properties. By focusing on minimal models, we avoid overestimating the adversary's power while capturing the essential attack conditions, thereby enabling a modular analysis of security requirements in terms of necessary and sufficient adversarial capabilities. However, the concept of a minimal adversary model is also non-unique in one critical aspect: for a given protocol, a security property, and a fixed set of atomic adversarial capabilities, multiple distinct minimal adversary models may exist. However, each minimal adversary model possesses uniqueness in its atomic capabilities. Even if a minimal model does not contain or is not fully contained within another minimal model, their atomic capabilities may partially overlap through non-empty intersections. Thus one needs the comprehensive identification of all minimal offensive adversary models to achieve an unambiguous characterization of the security properties in terms of atomic adversarial capabilities. Only through this full enumeration one can precisely express security properties as combinations of atomic adversarial capabilities relying on the irreducible nature of each minimal offensive adversary model.

This allows us to define a unique and irreducible formula for each security property given a protocol model and a set of atomic capabilities.

Definition 4.4.5 (Security Formula)**.** *Given a protocol model $\mathcal{P}$, a set of atomic capabilities Γ and a security property φ, a* security formula *is the logical disjunctions of all the minimal offensive adversary models $\overset{\bullet}{\mathcal{S}}_{i\mathcal{P},\Gamma,\varphi}$, defined as:*

$$\bigvee_{i=1}^{k} \overset{\bullet}{\mathcal{S}}{}^{\wedge}_{i\mathcal{P},\Gamma,\varphi} = \mathcal{S}^{\wedge}_{1\,\mathcal{P},\Gamma,\varphi} \vee \ldots \vee \mathcal{S}^{\wedge}_{k\,\mathcal{P},\Gamma,\varphi}$$

where k is the number of all minimal offensive adversary models.

The security formula literally states that the considered the security property φ (for the protocol model $\mathcal{P}$ and considering the set of atomic capabilities Γ) is violated if and only if the attacker has the atomic capabilities in $\overset{\bullet}{\mathcal{S}}_{1\mathcal{P},\Gamma,\varphi}$ or $\overset{\bullet}{\mathcal{S}}_{2\mathcal{P},\Gamma,\varphi}$ or … or $\overset{\bullet}{\mathcal{S}}_{k\mathcal{P},\Gamma,\varphi}$. Let us consider an example of a case study in order to put the later definitions into practice, find the minimal offensive models, and thus the security fomulas.

Example:

Consider the simplified Needham-Schroeder protocol (NS) as described in Chapter 1, and which is specified as follows:

$$\begin{array}{lcll} \text{A} & \to & \text{B} & : \ \mathsf{enc}((nA, \mathsf{pk}(skA)), \mathsf{pk}(skB)) \\ \text{B} & \to & \text{A} & : \ \mathsf{enc}((nA, nB), \mathsf{pk}(skA)) \\ \text{A} & \to & \text{B} & : \ \mathsf{enc}(nB, \mathsf{pk}(skB)) \end{array}$$

Alice and Bob have respectively $(skA, \mathsf{pk}(skA))$ and $(skB, \mathsf{pk}(skB))$ as secret and public key pairs. Alice generates a nonce nA and sends it along with her public key encrypted with Bob's public key. Upon receiving the message from Alice, Bob decrypts the message and retrieves nA, then generates a nonce nB and sends both nonces encrypted with the public key of Alice. Alice receives the message from Bob, retrieves nB and encrypts it with Bob's public key, and sends the ciphertext to Bob. The Needham-Schroeder protocol is meant to guanrentee *mutual authentication* between Alice and Bob.

Let $\Gamma = \{R_{skA}, R_{skB}, R_{nA}, R_{nB}\}$ where R_i refers to the case when i is generated or computed honestly yet it is revealed to the adversary at some time during the protocol's execution. We model the protocol in ProVerif, and all the adversarial capabilities are modeled as processes and annotated with the corresponding events. The ProVerif model is depicted in Figure 4.7. For example, to model R_{skA}, we create a process **RevealskA** which takes skA as argument, we define the event **RevskA** which takes pkA as argument, and we output skA on the public channel. As depicted in Figure 4.8, this process is placed in parallel in the main process. Whenever the key skA is generated, it can be revealed at any time to the attacker.

Let us consider authentication properties and let $\varphi_{Aut_{AB}}$ and $\varphi_{Aut_{BA}}$ be the authentication of A to B and of B to A respectively. We express the authentication properties as basic correspondence assertions expressing *weak agreement* as specified by [Low97]. For instance, we model $\varphi_{Aut_{BA}}$ as follows: for every execution trace of the protocol, if the event **endAparam**`(x)` is reached, then there should have been an earlier occurrence of the event **beginAparam**`(x)`. Put differently, if Alice believes that she has completed the protocol with Bob (the event **endAparam**`(x)` is placed at the end of the process of Alice and raised with her public key in Figure 4.7), then Bob has previously been running the protocol with Alice (the event **beginAparam**`(x)` is placed at the begining of the process of Bob and raised with the public key of Alice in Figure 4.7).

Offensive Models. The property $\varphi_{Aut_{AB}}$ does not hold within NS, thus, according to Definition 4.4.3, we have $\overset{\circ}{\emptyset}_{\text{NS},\Gamma,\varphi_{Aut_{AB}}}$ which is also minimal. This means that the attacker does not need to have any of the atomic capabilities to attack the security property. The property $\varphi_{Aut_{BA}}$ holds within the NS protocol, thus we need to find the offensive adversary models according to Definition 4.4.3 from all the possible adversary models, and identify the minimal models breaking this property. According to Definition 4.4.2, an adversary model is any subset of $\Gamma = \{R_{skA}, R_{skB}, R_{nA}, R_{nB}\}$. Thus, the number of the adversary models that we need to check is $16 = 2^4$, since the cardinality of Γ is 4. We structure all adversary models as a lattice ordered by set inclusion, where each node represents a distinct set of adversarial capabilities. This partial order captures containment relationships between adversary models. Such a representation enables systematic analysis of minimal adversarial requirements while preserving hierarchical relationships across the entire adversary models. The lattice corresponding to our running example is depicted in Figure4.6. The offensive adversary models are highlighted in dark blue. This representation directly mirrors a ProVerif model, where the main process, participant processes (A and B), and only the reveal processes specified in the lattice node are included. We write a single authentication query. If the query is false, we validate the considered adversary model as an offensive one, which corresponds exactly to Definition 4.4.3. The lattice structure reveals that the sets $\{R_{skB}\}$ and $\{R_{nA}\}$ are the minimal offensive models. The disjunction of all the minimal offensive models gives the following security formula $R_{skB} \vee R_{nA}$, and it admits the following interpretation: authentication of B to A in the simplified Needham-Schroeder protocol is guaranteed if and only if neither of the following compromise conditions occurs: the reveal of A's nonce to the adversary, or the reveal of B's secret key to the adversary.

No need to test all nodes. We can perform an adaptive search strategy for identifying minimal offensive adversary models which enables significant optimization by exploiting the lattice structure. Beginning with a bottom-up search, we first evaluate only the four sets of cardinality 1 as potential minimal models. This phase eliminates the need to test their parent nodes in subsequent steps if they are minimal. In this running example, for any remaining sets of cardinality 2 or higher, testing just one $\{R_{skA}, R_{nB}\}$ suffices to determine all minimal models. This optimization works because the lattice hierarchy guarantees that any superset of an offensive model cannot itself be minimal, allowing us to prune large portions of the search space while maintaining complete coverage. The resulting approach reduces computational effort without sacrificing rigor, as the minimality condition naturally propagates through the lattice structure. This description is the core of Algorithm 3 which we used in our analysis.

Remark:

Unlike ProVerif, Tamarin offers the possibility to encode a single protocol model while analyzing different adversary scenarios through lemmas. This is achieved by disabling compromise rules using logical negations ("not" statements), allowing a comprehensive security reasoning within a unified framework. However, this approach is only feasible for trace properties and cannot be applied in equivalence mode.

Building upon our prior work that characterized attacker models capable of compromising the security property, we now pursue another investigation into the

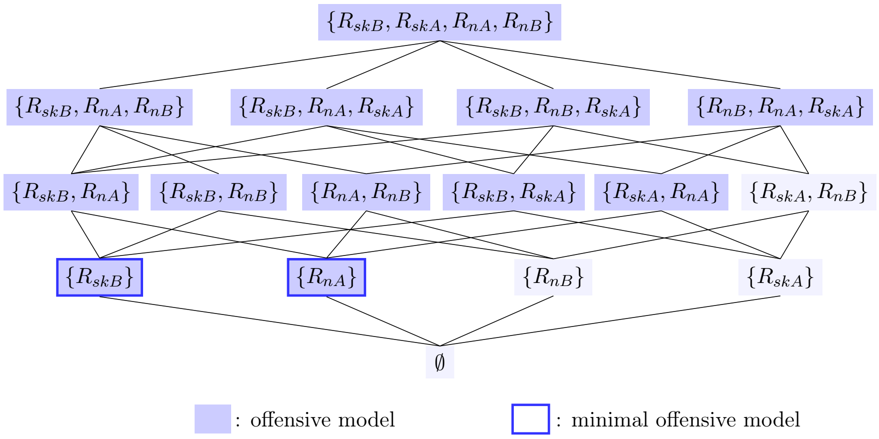


Figure 4.6: Lattice of offensive adversary models ordered by set inclusion for the Needham-Schroeder protocol [NS78].

main adversarial capabilities that are necessary conditions for successful attacks. This analysis aims to isolate and identify those atomic capabilities without which attacks cannot be mounted. This characteristic serves two important purposes: first, it formally gives a direct link from potential security breaches to specific atomic capabilities, and second, this reveals which information is most critical to secure by showing what would do the most damage if attackers got access to it. This approach thus not only describes possible compromise routes but also provides a way to measure which compromise matter most. This necessity-focused approach provides theoretical lower bounds on attacker capabilities. It identifies what is most important to secure which may help to focus protection efforts.

Definition 4.4.6 (Defensive Model)**.** *Let $\mathcal{S}$ be a subset of Γ. $\mathcal{S}$ is an* defensive model *for the security property φ within the protocol $\mathcal{P}$ with regard to the set of the atomic capabilities Γ, which we note $\overset{\triangleright}{\mathcal{S}}_{\mathcal{P},\Gamma,\varphi}$, if we have that $\mathcal{P}_{\mathcal{S}'} \vDash \varphi$ for all subsets $\mathcal{S}' \subseteq \Gamma \setminus \mathcal{S}$.*

A defensive model provides a critical security assurance by establishing that the violation of the specified security property necessarily requires the possession of one or more atomic capabilities defined within that model. If we have $\overset{\triangleright}{\mathcal{S}}_{\mathcal{P},\Gamma,\varphi}$ and $\mathcal{S} \subseteq \mathcal{S}'$ then $\overset{\triangleright}{\mathcal{S}'}_{\mathcal{P},\Gamma,\varphi}$, that is, adding atomic capabilities to the defensive model yields a defensive model. For instance, if $\mathcal{P}_{\emptyset} \vDash \varphi$ and $\mathcal{P}_{\Gamma} \nvDash \varphi$, then the set of all the atomic capabilities Γ is a defensive model. We need to tighten the bound regarding the atomic capabilities as our main purpose is to isolate minimal necessary atomic capabilities needed to mount attacks.

Definition 4.4.7 (Minimal Defensive Model)**.** *$\overset{\triangleright}{\mathcal{S}}_{\mathcal{P},\Gamma,\varphi}$ is said to be* minimal*, and refered to as $\overset{\blacktriangleright}{\mathcal{S}}_{\mathcal{P},\Gamma,\varphi}$, if it satisfies the following conditions:*

```
let RevealskA(skA: skey) =
  event RevskA(pk(skA));
  out(c, skA).

let RevealskB(skB: skey) =
  event RevskB(pk(skB));
  out(c, skB).

let RevealNa(Na: bitstring) =
  event RevnA(Na);
  out(c, Na).

let RevealNb(Nb: bitstring) =
  event RevnB(Nb);
  out(c, Nb).

let processA(pkB: pkey, skA: skey) =
  in(c, pkX: pkey);
  new Na: bitstring;
  event beginBparam(pkX);
  ((out(c, aenc((Na, pk(skA)), pkX));
  in(c, m: bitstring);
  let (=Na, Nb: bitstring) = adec(m, skA) in
  out(c, aenc(Nb, pkX));
  if pkX = pkB then
  event endAparam(pk(skA))) | (RevealNa(Na))).

let processB(pkA: pkey, skB: skey)=
  in(c, m: bitstring);
  let (Ny: bitstring, pkY: pkey) = adec(m, skB) in
  new Nb: bitstring;out(c, Nb);
  event beginAparam(pkY);
  ((out(c, aenc((Ny, Nb), pkY));
  in(c, m3: bitstring);
  if (Nb = adec(m3, skB)) then
  if pkY = pkA then
  event endBparam(pk(skB))) | (RevealNb(Nb))).
```

Figure 4.7: Protocol model of the Needham-Schroeder in ProVerif.

- $\mathcal{S} = \emptyset$, *or*
- $\mathcal{S} \neq \emptyset$ *and* $\forall s \in \mathcal{S}$, *the set* $\mathcal{S} \setminus \{s\}$ *is not a defensive model for* φ *within the protocol* $\mathcal{P}$ *with regard to the set of atomic capabilities* Γ.

Remark:

```
process
(
  new skA: skey; let pkA = pk(skA) in out(c, skA);
  new skB: skey; let pkB = pk(skB) in out(c, pkB);
  ((!processA(pkB, skA)) | (!processB(pkA, skB)) |
  RevealskA(skA) | RevealskB(skB))
)
```

Figure 4.8: Main process of the Needham-Schroeder in ProVerif.

If a property does not hold regardless of the adversary capabilities, then there exists no defensive model, and in particular no minimal defensive model. Thus, an empty minimal offensive adversary model corresponds to the Dolev-Yao attacker.

We now demonstrate the practical use of this definition through our running example, focusing on the computation of minimal defensive models.

Example:

Continuing our running example, consider the ProVerif model is given in Figures 4.7 and 4.8.

Defensive Models. Again, the property $\phi_{Aut_{AB}}$ does not hold within the Needham-Schroeder protocol [Low97] ($\mathrm{NS}_{\emptyset} \nvDash \phi_{Aut_{AB}}$), thus, there is no defensive model. The property $\phi_{Aut_{BA}}$ is satisfied within NS and in order to find the minimal defensive models, we needs the verification of every subset within Γ. First, each subset should be evaluated to determine whether it constitutes a valid defensive model according to our definition. For the subsets that satisfy the defensive model conditions, we must further verify whether they are minimal configurations with respect to set inclusion. This comprehensive analysis ensures the complete identification of all minimal defensive models present in our model. Our verification adopts a query-based research approach with a single protocol model containing all the compromise processes with Γ. The lattice corresponding to defensive models is depicted in Figure 4.10, each ProVerif query (presented in Figure4.9) corresponds to a node in the lattice. We find the following defensive models: $\{\mathcal{S}_1 = \{R_{skB}, R_{nA}\}, \mathcal{S}_2 = \{R_{skB}, R_{nA}, R_{skA}\}, \mathcal{S}_3 = \{R_{skB}, R_{nA}, R_{nB}\}, \Gamma\}$ (which correponds to the true queries in Figure 4.9). It is clear that $\overset{\blacktriangleright}{\mathcal{S}}_{1\mathrm{NS},\Gamma,\phi_{Aut_{BA}}}$ since we have that $\mathcal{S}_1 \subseteq \mathcal{S}_2$, $\mathcal{S}_1 \subseteq \mathcal{S}_3$ and $\mathcal{S}_1 \subseteq \Gamma$.

The first query in Figure 4.9 corresponds to the authentication property $\phi_{Aut_{BA}}$ expressed as the usual correspondence assertions and ProVerif finds that the property is not satisfied when the attacker has all the atomic capabilities in Γ ($\mathrm{NS}_{\emptyset} \vDash \phi_{Aut_{BA}}$ and $\mathrm{NS}_{\Gamma} \nvDash \phi_{Aut_{BA}}$). The first true query corresponds to the following: for every execution trace of the protocol, either the property is satisfied or the reveal of nA is executed or the reveal of skB is executed. In other terms, if the property is not satisfied than either the attacker has nA or skB. The set $\{R_{nA}, R_{skB}\}$ satisfies Definition 4.4.6, that is $\{R_{nA}, R_{skB}\}$ is a defensive model in this case. The same reasoning applies to all the other true queries in Figure 4.9. In this example, $|\Gamma| = 4$ so we needed to verify all the subsets within Γ, that is $2^4 = 16$ queries which corresponds to the nodes of the lattice depicted in Figure 4.10. Therefore, to sum up, in order to find the set of the defensive models for a protocol $\mathcal{P}$, the security property φ, and the set of the atomic compromises, we need one single input modeling the protocol and the atomic adversarial capabilities, a set of queries of the form: *property verified or the attacker has the atomic capability a or the attacker has the atomic capability b or ...*, and retrieve the queries which have been proven to be true. The minimal models are the smallest ones in the sense of set inclusion. However, one can argue that since $\mathcal{S}_1 \subseteq \mathcal{S}_2$ and $\mathcal{S}_1$ is a defensive model then $\mathcal{S}_2$ is also a defensive model. Put differently, since the query corresponding to $\mathcal{S}_1$ is true, then the one corresponding to $\mathcal{S}_2$ should also be true, and we do not need to double-check it. If we were checking the queries step-by-step, we would not need to verify this other true query in this case since $\mathcal{S}_1$'s query already covers it. Algorithm 4 is based on this reasoning. The conjunction of all the minimal defensive adversary models is given by the formula $R_{nA} \vee R_{skB}$ which literally means that the authentication of B to A holds unless the secret key of B is revealed or the nonce generated by A is revealed. Note that this formula is equivalent to the security formula obtained previously through the disjunction of minimal offensive adversary models.

We discuss two points that naturally arise from the previous examples.

Usefulness of both approaches. While both approaches produce equivalent final formulas, they differ significantly in methodology. The minimal defensive models approach lends itself naturally to query-based verification in ProVerif, enabling results to be obtained from a single model without modifying process definitions or adversarial capability rules. This is particularly advantageous for reducing the protocol verifier's preprocessing overhead during multiple executions. Notably, the model remains consistent even when employing adaptive procedures requiring multiple file executions. However, this approach has inherent limitations as certain security properties (such as equivalence-based properties) and adversarial capabilities fall outside its scope, requiring alternative non-query-based methods with existing protocol's verifiers. Conversely, the offensive adversary models approach can also operate as query-based when the query language

```
--------------------------------------------------------------
Verification summary:

Query event(endAparam(xA)) ==> event(beginAparam(xA)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevskA(xA)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevskB(xB)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnA(nA)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnB(nB)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnA(nA)) || event(RevnB(nB)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnA(nA)) || event(RevskA(xA)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnA(nA)) || event(RevskB(xB)) is true.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnB(nB)) || event(RevskA(xA)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnB(nB)) || event(RevskB(xB)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevskB(xB)) || event(RevskA(xA)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnA(nA)) || event(RevnB(nB)) || event(RevskA(xA)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnA(nA)) || event(RevnB(nB)) || event(RevskB(xB)) is true.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnA(nA)) || event(RevskB(xB)) || event(RevskB(xB)) is true.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnB(nB)) || event(RevskB(xB)) || event(RevskB(xB)) is false.

Query event(endAparam(xA)) ==> event(beginAparam(xA)) || event(RevnA(nA)) || event(RevnB(nB)) || event(RevskB(xB)) || event(RevskB(xB)) is true.

--------------------------------------------------------------
```

Figure 4.9: Outcome of ProVerif's queries.

offers sufficient expressiveness. For instance, Tamarin permits rule deactivation for specific adversarial capabilities within lemmas via negation, while ProVerif lacks this flexibility, requiring direct model modifications instead.

In our subsequent analysis of security protocols, we used a hybrid approach that combines both verification methodologies. To ensure the validity of our analysis, we prove that either method can be employed to obtain security formulas.

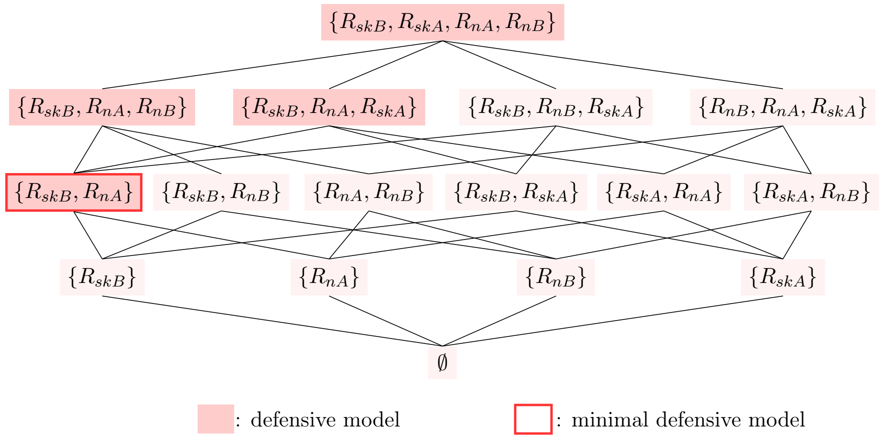


Figure 4.10: Lattice of defensive models for the Needham-Schroeder protocol ordered by set inclusion.

Lemma 4.4.1 (Non-Empty Intersection)**.** *Let $\mathcal{P}$ be a protocol model, φ a security property, and Γ a set of atomic capabilities. For all non-empty sets $\overset{\triangleright}{\mathcal{X}}_{\mathcal{P},\Gamma,\varphi}$ and $\overset{\circ}{\mathcal{Y}}_{\mathcal{P},\Gamma,\varphi}$, we have that $\mathcal{X} \cap \mathcal{Y} \neq \emptyset$.*

Proof. Suppose that there exist non-empty $\overset{\triangleright}{\mathcal{X}}_{\mathcal{P},\Gamma,\varphi}$ and $\overset{\circ}{\mathcal{Y}}_{\mathcal{P},\Gamma,\varphi}$ such that $\mathcal{X} \cap \mathcal{Y} = \emptyset$. We have $\mathcal{Y} \subseteq \Gamma \setminus \mathcal{X}$, and per Definition 4.4.6, $\mathcal{P}_{\mathcal{Y}} \vDash \varphi$. This constradicts the fact that the set $\mathcal{Y}$ is an offensive model. □

Lemma 4.4.2. *For all $(y_1, \ldots, y_k) \in \overset{\bullet}{\mathcal{Y}}_{1\mathcal{P},\Gamma,\varphi} \times \ldots \times \overset{\bullet}{\mathcal{Y}}_{k\mathcal{P},\Gamma,\varphi}$, where $\{\overset{\bullet}{\mathcal{Y}}_{i\mathcal{P},\Gamma,\varphi}\}_{1 \leq i \leq k}$ is the set of all the different non-empty minimal offensive adversary models; $\{y_j\}_{1 \leq j \leq k}$ is a defensive model.*

Proof. Suppose that there exists $(y_1, \ldots, y_k)$ such that $\{y_i\}_{1 \leq i \leq k}$ is not a defensive model. Thus, there exists $\mathcal{S} \subseteq (\Gamma \setminus \{y_i\}_{1 \leq i \leq k})$ such that $\mathcal{P}_{\mathcal{S}} \nvDash \varphi$. Since $\mathcal{S}$ is an offensive adversary model, then there exists a minimal offensive adversary model $\mathcal{Y}_j$ such that $\mathcal{Y}_j \subseteq \mathcal{S}$. Thus, there exists a non-empty $y_j \in \mathcal{S} \cap \{y_i\}_{1 \leq i \leq k}$ which is absurd because $\mathcal{S} \subseteq (\Gamma \setminus \{y_i\}_{1 \leq i \leq k})$. □

Corollary 4.4.1. *Given a minimal defensive model $\overset{\blacktriangleright}{\mathcal{X}}_{\mathcal{P},\Gamma,\varphi}$, there exists $(y_1, \ldots, y_k)$ $\in \overset{\bullet}{\mathcal{Y}}_{1\mathcal{P},\Gamma,\varphi} \times \ldots \times \overset{\bullet}{\mathcal{Y}}_{k\mathcal{P},\Gamma,\varphi}$ such that $\overset{\blacktriangleright}{\mathcal{X}}_{\mathcal{P},\Gamma,\varphi} = \{y_i\}_{1 \leq i \leq k}$, where $\{\overset{\bullet}{\mathcal{Y}}_{i\mathcal{P},\Gamma,\varphi}\}_{1 \leq i \leq k}$ is the set of all the non-empty minimal offensive adversary models.*

Proof. Per Lemma 4.4.1, we have that the intersection between a non-empty defensive model and a non-empty offensive adversary model is non-empty. In particular, the intersection between the non-empty minimal defensive adversary model $\overset{\blacktriangleright}{\mathcal{X}}_{\mathcal{P},\Gamma,\varphi}$ and each of the non-empty minimal offensive adversary models $\overset{\bullet}{\mathcal{Y}}_{i\mathcal{P},\Gamma,\varphi}$ is not empty. Hence, there exists $(y_1, \ldots, y_k) \in \overset{\bullet}{\mathcal{Y}}_{1\mathcal{P},\Gamma,\varphi} \times \ldots \times \overset{\bullet}{\mathcal{Y}}_{k\mathcal{P},\Gamma,\varphi}$ such that $\{y_i\}_{1 \leq i \leq k} \subseteq \overset{\blacktriangleright}{\mathcal{X}}_{\mathcal{P},\Gamma,\varphi}$. Per Lemma 4.4.2, the set $\{y_i\}_{1 \leq i \leq k}$ is a defensive model. However, $\overset{\blacktriangleright}{\mathcal{X}}_{\mathcal{P},\Gamma,\varphi}$ is a minimal defensive model and cannot include another defensive model but itself, thus $\overset{\blacktriangleright}{\mathcal{X}}_{\mathcal{P},\Gamma,\varphi} = \{y_i\}_{1 \leq i \leq k}$. □

Lemma 4.4.3. *For all $(x_1, \ldots, x_k) \in \overset{\blacktriangleright}{\mathcal{X}}_{1\mathcal{P},\Gamma,\varphi} \times \ldots \times \overset{\blacktriangleright}{\mathcal{X}}_{k\mathcal{P},\Gamma,\varphi}$, where $\{\overset{\blacktriangleright}{\mathcal{X}}_{i\mathcal{P},\Gamma,\varphi}\}_{1 \leq i \leq k}$ is the set of all the non-empty minimal defensive models; $\{x_j\}_{1 \leq j \leq k}$ is an offensive model.*

Proof. Suppose that there exists $(x_1, \ldots, x_k) \in \overset{\blacktriangleright}{\mathcal{X}}_{1\mathcal{P},\Gamma,\varphi} \times \ldots \times \overset{\blacktriangleright}{\mathcal{X}}_{k\mathcal{P},\Gamma,\varphi}$ such that $\mathcal{X} = \{x_i\}_{1 \leq i \leq k}$ is not an offensive adversary model. That is, for all $j \in \{1, \ldots, k'\}$, there exists $y_j \in \overset{\bullet}{\mathcal{Y}}_{j\mathcal{P},\Gamma,\varphi}$ and $y_j \notin \mathcal{X}$, where k' is the number of all the different non-empty minimal offensive models. Per Lemma 4.4.2, the set $\{y_j\}_{1 \leq j \leq k'}$ is a defensive model, that is, containing a minimal non-empty defensive model, i.e., there exists $l \in \{1, \ldots, k\}$ such that $\overset{\blacktriangleright}{\mathcal{X}}_{l\mathcal{P},\Gamma,\varphi} \subseteq \{y_j\}_{1 \leq j \leq k'}$. This is absurd since $y_j \notin \mathcal{X}$ for all $j \in \{1, \ldots k'\}$. □

Corollary 4.4.2. *Given a minimal offensive model* $\overset{\bullet}{\mathcal{Y}}_{\mathcal{P},\Gamma,\varphi}$*, there exists* $(x_1,\ldots,x_k) \in \overset{\blacktriangleright}{\mathcal{X}}_{1\mathcal{P},\Gamma,\varphi}\times\ldots\times\overset{\blacktriangleright}{\mathcal{X}}_{k\mathcal{P},\Gamma,\varphi}$*, where* $\{\overset{\blacktriangleright}{\mathcal{X}}_{i\mathcal{P},\Gamma,\varphi}\}_{1\leq i\leq k}$ *is the set of all the different non-empty minimal defensive models, such that* $\overset{\bullet}{\mathcal{Y}}_{\mathcal{P},\Gamma,\varphi} = \{x_i\}_{1\leq i\leq k}$*.*

Proof. Per Lemma 4.4.1, we have that for all non-empty $\overset{\bullet}{\mathcal{Y}}_{\mathcal{P},\Gamma,\varphi}$, there exists $(x_1,\ldots,x_k) \in \overset{\blacktriangleright}{\mathcal{X}}_{1\mathcal{P},\Gamma,\varphi}\times\ldots\times\overset{\blacktriangleright}{\mathcal{X}}_{k\mathcal{P},\Gamma,\varphi}$ such that $\{x_i\}_{1\leq i\leq k} \subseteq \overset{\bullet}{\mathcal{X}}_{\mathcal{P},\Gamma,\varphi}$. Per Lemma 4.4.3, the set $\{x_i\}_{1\leq i\leq k}$ is an offensive model. Since $\overset{\bullet}{\mathcal{Y}}_{\mathcal{P},\Gamma,\varphi}$ is a minimal offensive model, then $\overset{\bullet}{\mathcal{X}}_{\mathcal{P},\Gamma,\varphi} = \{x_i\}_{1\leq i\leq k}$. □

Theorem 4.4.1. *The conjunction of all non-empty minimal defensive models yield a security formula:*

$$\bigvee_{j=1}^{k}\overset{\bullet\ \wedge}{\mathcal{Y}}_{j\mathcal{P},\Gamma,\varphi} = \bigwedge_{i=1}^{k'}\overset{\blacktriangleright\ \vee}{\mathcal{X}}_{i\mathcal{P},\Gamma,\varphi}$$

where k *and* k' *are the number of all minimal non-empty offensive adversary models and all non-empty minimal defensive models respectively.*

Proof.

$$\bigwedge_{i=1}^{k'}\overset{\blacktriangleright\ \vee}{\mathcal{X}}_{i\mathcal{P},\Gamma,\varphi} = \bigvee_{(x_1,\ldots,x_{k'})\in\mathcal{X}_1\times\ldots\times\mathcal{X}_{k'}} x_1\wedge\ldots\wedge x_{k'}$$

Per Lemma 4.4.3, each subset $\{x_1,\ldots,x_{k'}\}$ figuring in the formula, is an offensive model. Moreover, according to Corollary 4.4.2, given a $\overset{\bullet}{\mathcal{Y}}_{\mathcal{P},\Gamma,\varphi}$, there exists $(x_1,\ldots,x_{k'}) \in \overset{\blacktriangleright}{\mathcal{X}}_{1\mathcal{P},\Gamma,\varphi}\times\ldots\times\overset{\blacktriangleright}{\mathcal{X}}_{k'\mathcal{P},\Gamma,\varphi}$ such that $\{x_1,\ldots,x_{k'}\}$ is a minimal offensive model. Since we have all the combination of $(x_1,\ldots,x_{k'}) \in \mathcal{X}_1\times\ldots\times\mathcal{X}_{k'}$, there must be minimal configurations of $(x_1,\ldots,x_{k'})$ expressing minimal offensive models, which will subsume all the other non-minimal offensive models (since we have a logical `or` between the conjunctions of capabilities). Consequently, the remaining configurations correspond precisely to minimal defensive models. This establishes the equality with the security formula.

Conversely, the security formula can also be written as the conjunction of minimal defensive models. We have that:

$$\bigvee_{i=1}^{k}\overset{\bullet\ \wedge}{\mathcal{Y}}_{i\mathcal{P},\Gamma,\varphi} = \bigwedge_{(y_1,\ldots,y_k)\in\mathcal{Y}_1\times\ldots\times\mathcal{Y}_k} y_1\vee\ldots\vee y_k$$

Per Lemma 4.4.2, each subset $\{y_1, \ldots, y_k\}$ figuring in the formula, is a defensive model and according to Corollary 4.4.1, given a $\overset{\blacktriangleright}{\mathcal{X}}_{\mathcal{P},\Gamma,\varphi}$, there exists $(y_1, \ldots, y_k) \in \overset{\bullet}{\mathcal{Y}}_{1\mathcal{P},\Gamma,\varphi} \times \ldots \times \overset{\bullet}{\mathcal{Y}}_{k\mathcal{P},\Gamma,\varphi}$ such that $\{y_1, \ldots, y_k\}$ is a minimal defensive model. Since we have all the combination of $(y_1, \ldots, y_k) \in \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_k$, there must be minimal configurations of $(y_1, \ldots, y_k)$ expressing minimal defensive models and subsuming all other non-minimal defensive models.

□

Remark:

Deriving security formulas through a query-based approach can be a bit tricky and subtle. The following example illustrates these subtleties in practice.

Example:

Let us consider the toy protocol described in Figure 4.11 in the applied Π-Calculus à la Sapic$^+$. The protocol is executed between two parties, where the initiator transmits one message to the responder. It executes as follows: Both parties share a pre-shared key *psk*. The initiator generates a nonce n and a session key k, then sends $\mathsf{enc}\{k\}_{pkR}$ and $\mathsf{enc}\{\langle psk, n\rangle\}_k$ to the receiver. The receiver decrypts $\mathsf{enc}\{k\}_{pkR}$ with their private key skR to recover k, then decrypts $\mathsf{enc}\{\langle psk, n\rangle\}_k$ and verifies that the decrypted pre-shared key matches the expected psk to authenticate the initiator. If valid, the nonce n is accepted.

Let $\Gamma = \{R_{skR}, R_n, R_k, R_{psk}\}$ where R_i stands for the reveal of x to the attacker. We consider the *non-injective agreement* property as specified in [Low97], and which we fomulate as follows: for every execution trace of the protocol if the event **Rreceive**(psk, pkR, n, k) is reached, then there should have been an earlier occurrence of the event **Isend**(psk, pkR, n, k). In the absence of any atomic compromise, the authentication property is verified. However, when searching for the security formula via a query-based approach, the results given by ProVerif are as depicted in the Figure 4.12[a]. All queries are false! The key subtlety arises from the event parameterization in the queries. For instance, if we change the parameter in the event **eRevk** by replacing k_3 with $\mathbf{k'}$, we obtain the results depicted in Figure 4.13. Therefore, the security formula corresponding to the week agreement property for our toy protocol is: $R_k \vee R_{psk} \vee R_{skR}$. The key insight is that query-based security verification requires two critical conditions. First, the security property must hold when no atomic capabilities are present, otherwise, the analysis becomes meaningless, and searching for minimal configurations loses purpose. Second, the set of all the atomic capabilities Γ should be a defensive model, meaning that the query that encompasses all possible atomic capabilities within Γ should be verified.

[a] Note that event names in this figure include an 'e_' prefix, compared to the events presented in Figure 4.11 since we translated the *sapic* file to ProVerif, this translation implies the addition of this prefix to these events.

```
let Revealn(n) =
  event Revn(n);
  out(n)

let Revealk(k) =
  event Revk(k);
  out(k)

let Revealsk(sk) =
  event Revsk(pk(sk));
  out(sk)

let Revealpsk(psk) =
  event Revpsk(psk);
  out(psk)

let Initiator(psk, pkR) =
  new ~k;  new ~n;  (
  let ck = aenc(~k, pkR) in
  let c = senc(<psk, ~n>, ~k) in
  event Isend(psk, pkR, ~n, ~k);  out(<ck, c>)
  )| Revealn(~n) | Revealk(~k))

let Responder(psk, skR) =
  in(<ck, c>);
  let k = adec(ck, skR) in
  let <=psk, n> = sdec(c, k) in
  event Rreceive(psk, pk(skR), n, k).

process:
  new ~psk;  new ~skR; let pkR = pk(~skR) in out(pkR);
 (Initiator(~psk, pkR) | Responder(~psk, ~skR) | Revealsk(~skR)
| Revealpsk(~psk))
```

Figure 4.11: A toy protocol example in Sapic$^+$.

In summary, the analyses conducted in the following sections build upon the foundations derived in the preceding discussion. We establish formal specifications for each verified security property by characterizing both necessary and sufficient conditions for compromise. We primarily use minimal offensive models

```
--------------------------------------------------------------
Verification summary:

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j
      is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevk(k_3))@j
      is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevn(n_3))@j
      is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevsk(pkR))@j
      is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevpsk(psk_4))@j
      is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevk(k_3))@j
      || event(eRevn(n_3))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevk(k_3))@j
      || event(eRevsk(pkR))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevk(k_3))@j
      || event(eRevpsk(psk_4))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevn(n_3))@j
      || event(eRevsk(pkR))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevn(n_3))@j
      || event(eRevpsk(psk_4))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevpsk(psk_4))@j
      || event(eRevsk(pkR))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevpsk(psk_4))@j
      || event(eRevsk(pkR))@j || event(eRevn(n_3))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevpsk(psk_4))@j
      || event(eRevsk(pkR))@j || event(eRevk(k_3))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevn(n_3))@j
      || event(eRevpsk(psk_4))@j || event(eRevk(k_3))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevn(n_3))@j
      || event(eRevsk(pkR))@j || event(eRevk(k_3))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevk(k_3))@j
      || event(eRevn(n_3))@j || event(eRevsk(pkR))@j || event(eRevpsk(psk_4))@j is false.

--------------------------------------------------------------
```

Figure 4.12: Outcome of ProVerif's queries for Example 4.4.

for analyzing equivalence properties and public key distribution compromises, while relying on minimal defensive models for trace properties and atomic capabilities involving data reveal. The security formulas were systematically derived using ProVerif as a backend through the execution of Algorithm 4 for finding minimal defensive models, and Algorithm 3 for finding offensive adversary models. Both algorithms work as follows: We first generate the full lattice $\mathcal{L}_\Gamma = (\mathcal{L}_{ij})$ ordered by set inclusion, and such that $\mathcal{L}_{ij}$ correponds to the set of cardinality i, and j ranges from 1 (the initial node corresponding to the first set of cardinality i) to n_i (the total number of sets with cardinality i). We begin a bottom-up search by testing all nodes of cardinality 1. For each evaluated node, we eliminate it from the lattice, and if it is found to be minimal, we also remove it along with all its parent nodes. We then proceed to nodes of cardinality 2, repeating the same process for all remaining nodes. The search terminates when no nodes remain in the lattice. We note that in the query-based search, all remaining nodes of the same cardinality are tested within the same files. Sub-

```
--------------------------------------------------------------
Verification summary:

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j
      is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevk(k'))@j
      is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevn(n_3))@j
      is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevsk(pkR))@j
      is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevpsk(psk_4))@j
      is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevk(k'))@j
      || event(eRevn(n_3))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevk(k'))@j
      || event(eRevsk(pkR))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevk(k'))@j
      || event(eRevpsk(psk_4))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevn(n_3))@j
      || event(eRevsk(pkR))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevn(n_3))@j
      || event(eRevpsk(psk_4))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevpsk(psk_4))@j
      || event(eRevsk(pkR))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevpsk(psk_4))@j
      || event(eRevsk(pkR))@j || event(eRevn(n_3))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevpsk(psk_4))@j
      || event(eRevsk(pkR))@j || event(eRevk(k'))@j is true.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevn(n_3))@j
      || event(eRevpsk(psk_4))@j || event(eRevk(k'))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevn(n_3))@j
      || event(eRevsk(pkR))@j || event(eRevk(k'))@j is false.

Query event(eRreceive(psk_4,pkR,n_3,k_3))@i ==> event(eIsend(psk_4,pkR,n_3,k_3))@j || event(eRevk(k'))@j
      || event(eRevn(n_3))@j || event(eRevsk(pkR))@j || event(eRevpsk(psk_4))@j is true.

--------------------------------------------------------------
```

Figure 4.13: Outcome of ProVerif's queries for Example with slight change in the parameters 4.4.

sequent verification in Tamarin is used to verify the security formula expressed as a single lemma. Tamarin's `diffie-hellman` builtin, which treats modular exponentiation as an AC (Associative-Commutative) symbol, provides more accurate cryptographic modeling and stronger algebraic reasoning capabilities for exponential operations.

4.5 Symbolic Analysis of WireGuard with Sapic$^+$

In this section, we propose a detailed formal model of WireGuard in the applied Π-Calculus, the language used by the framework Sapic$^+$ [CJKK22].

4.5.1 Adversary Model, Security Formulas

We consider 34 security properties; 4 authentication properties: agreement on `RecHello` message (from the Responder to the Initiator), agreement on the first

Algorithm 3 Algorithm for minimal offensive models.

Input: $\mathcal{P}$ a protocol, φ a security property such that $\mathcal{P}_{\emptyset} \vDash \varphi$, Γ a set of atomic capabilities of cardinality n and $\mathcal{L}_{\Gamma} = (\mathcal{L}_{ij})$ the lattice for Γ ordered by set inclusion.

Output: Set of minimal offensive models $\mathcal{S}$

```
$\mathcal{S} \leftarrow \emptyset$
for i = 1 to n do
    $n_i \leftarrow$ nb of subsets of cardinality $i$
    for j = 1 to $n_i$ do
        if $\mathcal{P}_{\mathcal{L}_{ij}} \nvDash \varphi$ then
            $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{L}_{ij}$
            remove parents of $\mathcal{L}_{ij}$ from $\mathcal{L}_{\Gamma}$
        elif
            remove $\mathcal{L}_{ij}$ from $\mathcal{L}_{\Gamma}$
        end if
    end for
end for
```

Algorithm 4 Algorithm for minimal defensive models.

Input: $\mathcal{P}$ a protocol, φ a security property such that $\mathcal{P}_{\emptyset} \vDash \varphi$, Γ a set of atomic capabilities of cardinality n and $\mathcal{L}_{\Gamma} = (\mathcal{L}_{ii})$ the lattice for Γ ordered by set inclusion.

Output: Set of minimal defensive models $\mathcal{S}$

```
$\mathcal{S} \leftarrow \emptyset$
for i = 1 to n do
    $n_i \leftarrow$ nb of subsets of cardinality $i$
    for j = 1 to $n_i$ do
        $\varphi_{L_{ij}} \leftarrow \varphi \vee L_{ij}^{\vee}$
        if $\mathcal{P}_{\Gamma} \vDash \varphi_{L_{ij}}$ then
            $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{L}_{ij}$
            remove parents of $\mathcal{L}_{ij}$ from $\mathcal{L}_{\Gamma}$
        else
            remove $\mathcal{L}_{ij}$ from $\mathcal{L}_{\Gamma}$
        end if
    end for
end for
```

`TransData` message (from the Initiator to the Responder), agreement of next `TransData` messages (from the Initiator to the Responder and from the Responder to the Initiator); 12 secrecy properties: secrecy and Forward Secrecy (FS) of the session key before derivation (named k_6 in protocol description), from the Initiator's and the Responder's view, secrecy and FS of the derived keys (named C^i and C^r), from the Initiator's and the Responder's view; anonymity, for WireGuard with or without cookies.

For each security property, for each protocol version (with or without cookies), our adversary model implies up to $2^{21} = 2^{6+6+7+2} = \mathbf{2097152}$ cases of key compromises, as our adversary can:

- Read the Initiator (resp. the Responder) static private key u (resp. v), the Initiator (resp. the Responder) ephemeral private key x (resp. y), the pre-shared key psk, the ECDH pre-computation at some time during the protocol execution (2^6 cases).
- Read the Initiator (resp. the Responder) static private key u (resp. v), the Initiator (resp. the Responder) ephemeral private key x (resp. y), the pre-shared key psk, the ECDH pre-computation *after* the protocol execution (2^6 cases).
- Modify the Initiator (resp. the Responder) static private key u (resp. v), the Initiator (resp. the Responder) ephemeral private key x (resp. y), the pre-shared key psk, the ECDH pre-computation for Initiator or Responder (2^7 cases) at some time during the protocol execution.
- Modify the Initiator's static public key U (resp. the Responder static public key V) distribution (2^2 cases).

Remark:

The expressions "the adversary can read d" or "d is revealed to the adversary" is equivalent to the atomic capability R_d when d is generated or computed honestly yet revealed to the adversary.

We first analyze WireGuard without cookies. Our main idea is to capture key modification and the compromise of the public keys distribution through different *models* searching for minimal offensive models, and key reveal through *queries* based on Algorithms 3 and 4. We use Sapic$^+$ to generate all the ProVerif files and we use ProVerif to derive the security formulas for each security property. Through this methodological synthesis, we derive a single Tamarin lemma that encapsulates our security formula. This lemma is then evaluated using a single generated Tamarin input file, enabling the verification of the security formula with a more precise equational theory (the Diffie-Hellman builtin of Tamarin).

We consider a first set of adversarial capabilities: key reveal at any time during the protocol execution (2^6 cases), key modifications (2^7 cases), and key distribution compromise (2^2 cases). This set is used to assess all agreement and secrecy properties.

Then, we consider a second set: key reveal after the protocol execution for static keys and the pre-shared key (with key reveal at any time during the protocol for the ephemeral keys and the randomness), and key distribution compromise (2^2 cases). This allows to capture precisely forward secrecy and also current WireGuard implementation, where a single configuration file contains both private static key and pre-shared key.

Finally, we consider anonymity with the atomic capabilities based on the reveal of data, which we do not combine with key modification nor key distribution modification.

Once all assessments are done for WireGuard without cookies, we obtain a set of compact security formulas for each security property. We then reuse directly these formulas for the model of WireGuard with cookies. We also reuse these formulas to assess the two fixes we propose as modifications of WireGuard to guarantee anonymity. The fixes result in equivalent security levels for both agreement and secrecy properties, producing identical minimal defensive and offensive models.

4.5.2 Agreement, Secrecy and Perfect Forward Secrecy

We model agreement as a trace property for the `RecHello` message, the first `TransData` message from the Initiator to the Responder and the next `TransData` messages which can be either from the Initiator to the Responder or from the Responder to the Initiator. We model the following notion: if a message has been received by a peer, then it must have been sent by the other peer. Using the notations from Section 4.2, we model key secrecy as a trace property for the keys k_6, C^i, C^r, from Initiator's and Responder's points of view.

Reference process

```
( (! Initiator(~ltkI, 'g'^~ltkI, 'g'^~ltkR, ~psk, ...))
  | (! Responder(~ltkR, 'g'^~ltkI, 'g'^~ltkR, ~psk, ...))
  | RevealPsk(~psk)
  | RevealLtki(~ltkI)
  | RevealLtkr(~ltkR) | RevealPre(~ltkI, ~ltkR) )

let Initiator(~ltkI, pkI, pkR, ~psk, ...) =
    ... new ~ekI; ... let pekI  = 'g'^~ekI in
    (...) | (RevealEki(~ekI))

let Responder(~ltkR, pkI, pkR, ~psk, ...) =
    ... new ~ekR; ... let pekR  = 'g'^~ekR in
    (...) | (RevealEkr(~ekR)) ...
```

Figure 4.14: SAPIC$^+$ main processes for the WireGuard protocol.

For these properties, we start with a *reference* model depicted in Figure 4.14, in which the generated keys in the main process are: the Initiator and the Responder static private keys, (∼`ltkI`, ∼`ltkR`) and the pre-shared key (∼`psk`). The keys are passed as arguments to two sub processes, `Initiator` and `Responder`. The process `Initiator` has the arguments ∼`ltkI` (its own static private key), `pkI = 'g'^`∼`ltkI` (its own static public key), `pkR = 'g'^`∼`ltkR` (the Responder's static public key), and ~psk (pre-shared key). The process Responder has similar arguments. Five processes model key compromise: RevealPsk for the pre-shared key, RevealLtki and RevealLtkr for the static keys, and RevealPre for the pre-computation, RevealEki and RevealEkr for the ephemeral keys. The Initiator and Responder processes are called in parallel with these compromise processes and replicated.

For FS properties, the methodology is the same. In addition, to capture temporal key compromise, we use the notion of **phase** in the generated ProVerif files: (RevealPsk(~psk)) is replaced with (phase 1: RevealPsk(~psk)). We add the same modification for RevealLtki and RevealLtkr.

Prior to presenting the analysis results, we restate the notation employed in the security formulas for the atomic capabilities examined in our analysis.

- R_x (resp. R_y) Initiator's (resp. Responder's) ephemeral key is revealed,
- M_x (resp. M_y) Initiator's (resp. Responder's) ephemeral key is modified,
- R_u (resp. R_v) Initiator's (resp. Responder's) static key is revealed,
- M_u (resp. M_v) Initiator's (resp. Responder's) static key is modified,
- R_u^* (resp. R_v^*) Initiator's (resp. Responder's) static key is revealed *after* protocol execution.
- R_s pre-shared key is revealed,
- M_s pre-shared key is modified,
- R_s^* pre-shared key is revealed *after* protocol execution.
- R_c pre-computation is revealed,
- M_i (resp. M_r) Initiator's (resp. Responder's) pre-computation is modified,
- D_u (resp. D_v) Initiator's (resp. Responder's) static key is compromised during initial distribution.

Results of the analysis. The results of our analysis for agreement and secrecy properties are depicted in Table 4.2. The security formulas for agreement on the RecHello message and for transport messages from the Responder to the Initiator are identical. When the Initiator receives and accepts the RecHello message, this leads to full authentication of the Responder by the Initiator and mutual agreement on the same keys, which in turn maintains the Responder's authentication toward the Initiator during subsequent communication. The minimal offensive adversary model $R_s \wedge R_v$ corresponds to an attacker having access to all static keys of the Responder which compromises the agreement property. In all the offensive models involving the Initiator's secret key u, compromising the agreement on the RecHello message requires additionally compromising the Initiator's ephemeral key x, e.g., $R_s \wedge R_u \wedge R_x$ is a minimal offensive adversary model. This condition provides a protection against key-compromise impersonation attacks.

The security formulas for the agreement on TransData messages from the Initiator to the Responder, are also identical. In fact, this security formula is symmetric to the security formula for the agreement on messages in the opposite direction (from the Responder to the Initiator), that is, up to key permutation (substituting the Initiator's keys with the Responder's keys) it remains the same formula. Consequently, the same conclusion applies to agreement on `TransData` messages sent by the Initiator. This symmetry in security formulas stems from the Diffie-Hellman computations performed bilaterally by peers. Such symmetry in the security guarantees will be a design challenge for the Post-Quantum WireGuard [HNS+21] when Diffie-Hellman is replaced with KEMs, and as we will see in the following chapter.

The security formulas for the secrecy on the session key from the Initiator's point of view and for the agreement on `RecHello` are identical when considering atomic capabilities at any time during the protocol execution. The same observation applies for the security formulas from the Responder's point of view. That is, agreement is guaranteed if and only if the derived key remains secret during the protocol's execution. The security formulas show that forward secrecy of session keys is guaranteed: static keys reveals only compromise secrecy when paired with ephemeral keys.

We further observe that across all security formulas if a minimal offensive model exists where an attacker can modify all static and ephemeral secret keys, then an equivalent minimal offensive model exists (for the same key set) where the secret keys are simply revealed to the attacker or when they are dishonestly generated. This effectively constrains the attacker's power, as what they can do the most is equivalent to when they have access to the keys during the protocol execution. In our results table, we exhibit this conclusion by reducing the security formulas to forms containing only the atomic capabilities corresponding to key reveal and the compromise of the distribution of public keys. For instance, the security formula for the agreement on the `RecHello` message, is reduced to the security formula $(D_v \wedge R_s) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x)$.

An attacker compromising public key distribution would only require the pre-shared key to break the protocol, i.e., $D_v \vee R_s$ and $D_u \vee R_s$ are minimal offensive models. Note that the WireGuard protocol does not handle public key exchange directly, though protocol designers recommend regular key rotation to guarantee sessions unlinkability. Consequently, public key exchange/distribution compromise represents a plausible scenario in this context, with the pre-shared key serving as the only defensive mechanism - though its use remains optional in the protocol specification. The pre-shared key constitutes a minimal defensive model in all security formulas. Thus, it requires robust protection rather than remaining merely optional.

We note that we included the ECDH precomputation compromise as an atomic capability R_c in our analysis since current implementations propose it as an optimization when the `InitHello` messages are received. The security formulas reveal that this introduces novel attack vectors through offensive adversary models involving the atomic capability R_c. For instance, $(R_c \wedge R_s \wedge R_x)$ and $(R_u \wedge R_s \wedge R_x)$ are both offensive models for the secrecy of the session key from the Initiator's point of view and for the agreement on the `RecHello` message. This means that an adversary with access to the ECDH pre-computation has as the same power as an adversary with access to the static private key. As explained in Section 4.3.2, this contradicts the ECDH property as an adversary with an access to the pre-computation should not be this powerful. We therefore recommend to remove this implementation optimization, and to compute the ECDH at the `InitHello` reception.

4.5.3 Anonymity

We model anonymity with observational equivalence in the following context: two Initiators, using their public keys U_1 and U_2, can establish a WireGuard session with a common Responder, using their public key V. The property is satisfied if an adversary, which has access to these public keys and to the exchanged messages, cannot tell which Initiator has established a session. We found that, opposed to initial claims in the original specification [Don17] and to the symbolic analysis with Tamarin [DM18], and in accordance with computational analysis with CryptoVerif [LBB19], this property is not satisfied: Figure 4.15 depicts an attack against anonymity, identified using our model with ProVerif. An Initiator, using their public key $U_* \in (U_1, U_2)$, establishes a session with Responder. They exchange a `RecHello` message whose 7^{th} field equals $\mathsf{MAC}(\mathsf{Hash}(\mathsf{M}, U_*), [2 \parallel \cdots \parallel \{\varnothing\}])$, where $[2 \parallel \cdots \parallel \{\varnothing\}]$ are the first 6 fields of `RecHello` message and M is a public constant. An adversary, which knows U_1 and U_2, can then compute $\mathsf{MAC}(\mathsf{Hash}(\mathsf{M}, U_1), [2 \parallel \cdots \parallel \{\varnothing\}])$, and $\mathsf{MAC}(\mathsf{Hash}(\mathsf{M}, U_2), [2 \parallel \cdots \parallel \{\varnothing\}])$ and assess which public key U_1 or U_2 has been used in the message authentication code by comparing them to the transmitted value $\mathsf{MAC}(\mathsf{Hash}(\mathsf{M}, U_*), [2 \parallel \cdots \parallel \{\varnothing\}])$. Finally, an adversary can distinguish between the two Initiators.

Anonymity in previous analyses. The analysis from [DM18] proposes a proof with Tamarin prover of a property named *identity hiding*, modeled as a trace property. As noticed in Section 4.3.1, their model does not include the real message authentication codes in `InitHello` and `RecHello` messages. Using the symbols from Section 4.5.2, anonymity has the minimal offensive models $R_u \vee R_v \vee R_x$. Similarly, the analysis from [GHS$^+$20] models and proves anonymity for `IKpsk2` with observational equivalence using the Tamarin prover. Anonymity for the Initiator has the offensive models $R_v \vee R_u$, and anonymity for the Responder has the minimal offensive models $R_v \vee R_u$. Finally, the analysis from [LBB19] describes the same attack as the one we identified, and proposes a

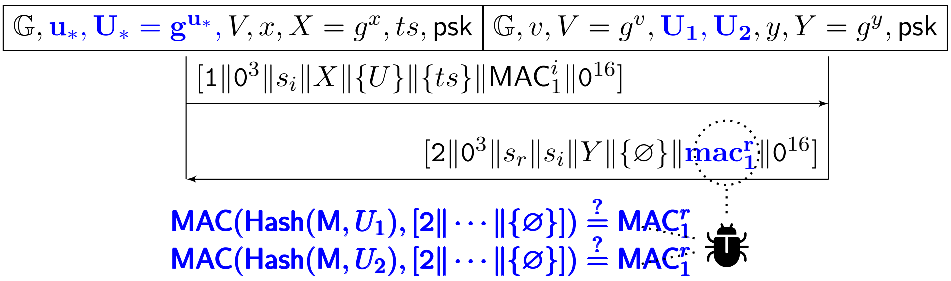

$\mathbb{G}, \mathbf{u}_*, \mathbf{U}_* = \mathbf{g}^{\mathbf{u}_*}, V, x, X = g^x, ts, \mathsf{psk}$
$\mathbb{G}, v, V = g^v, \mathbf{U_1}, \mathbf{U_2}, y, Y = g^y, \mathsf{psk}$
$[1 \| 0^3 \| s_i \| X \| \{U\} \| \{ts\} \| \mathsf{MAC}_1^i \| 0^{16}]$
$[2 \| 0^3 \| s_r \| s_i \| Y \| \{\varnothing\} \| \mathbf{mac_1^r} \| 0^{16}]$
$\mathsf{MAC}(\mathsf{Hash}(\mathsf{M}, U_1), [2 \| \cdots \| \{\varnothing\}]) \overset{?}{=} \mathsf{MAC}_1^r$
$\mathsf{MAC}(\mathsf{Hash}(\mathsf{M}, U_2), [2 \| \cdots \| \{\varnothing\}]) \overset{?}{=} \mathsf{MAC}_1^r$

Figure 4.15: Attack against Anonymity, where blue bold denotes attacker computation: attacker captures MAC_1 field from `RecHello` message, then compares with two possible values, computed with public keys $\mathbf{U_1}$ and $\mathbf{U_2}$.

proof of anonymity with CryptoVerif, for a modified version of the protocol without message authentication codes in the `InitHello` and the `RecHello` messages. This property has the minimal offensive model R_y. These three results do not directly apply to WireGuard as all do not consider message authentication codes in the two first messages, which is exactly the reasons why the adversary can break anonymity.

Proposed fixes. We propose fixes that ensure anonymity and do not change the messages. For this, we remark that the message authentication codes in `InitHello` and `RecHello` messages involve as keys only data potentially known by an adversary: the public value M and the public keys of the Initiator and the Responder, hence allowing to attack anonymity, as these authentication codes leak the used public key. To counter this attack, we propose a modification of the message authentication code computation to ensure the key for message authentication code is only known by the Initiator and the Responder. For this, we propose to use as either the value $\mathsf{Hash}(U \| g^{uv})$ or the value $\mathsf{Hash}(U \| \mathsf{psk})$ (instead of the value $\mathsf{Hash}(\mathsf{M} \| U)$ currently used) as the key for message authentication code in the `RecHello` message. We analyzed the modified protocol in the same context as before, and we prove that anonymity is guaranteed. For message authentication key $\mathsf{Hash}(U \| g^{uv})$, the corresponding security formula for anonymity is $R_u \vee R_v \vee R_x \vee R_c$; for message authentication key $\mathsf{Hash}(U \| \mathsf{psk})$, the security formula is $R_v \vee R_s \vee R_x$. We finally remark that we can confirm results from [GHS+20] as when removing MAC computations from `InitHello` and `RecHello` messages, the security formula for anonymity becomes $R_u \vee R_v \vee R_x \vee R_c$.

4.5.4 Performances

We evaluated our models on a dedicated server, equipped with 256 CPU cores running at 1.5 GHz. Agreement, secrecy and PFS are verified with ProVerif in around 15 minutes (for each property) and anonymity is verified with ProVerif in around 9 hours for the fix based on g^{uv}, and 2 hours for the fix based on the psk. We tested our result for trace properties (agreement, secrecy, PFS) with a lemma in Tamarin for a full version of the protocol and Tamarin confirmed that the properties are satisfied in around 5 hours. For anonymity, it is important to note that Sapic$^+$ does not currently provide support for the translation of equivalence properties into Tamarin. All of our files are publicly accessible and can be accessed online through a Gitlab repository [2].

4.5.5 Comparison with Previous Analyses

We compare our results with results on WireGuard [DM18] and on `IKpsk2` ([KNB19] and [GHS$^+$20]).

Comparison with [DM18]. This analysis only considers key reveals scenarios (without modification). Translating its results to our notations, these are: agreement on `RecHello` holds unless $R_s \wedge (R_v \vee (R_u \wedge R_x))$, agreement on first `TransData` message holds unless $(R_s \wedge (R_u \vee (R_v \wedge R_y)))$, secrecy holds unless $(R_s \wedge ((R_u \wedge R_x) \vee (R_v \wedge R_y))$, FS holds unless $(R^*_s \wedge ((R^*_u \wedge R_x) \vee (R^*_v \wedge R_y))$. For secrecy, the modeled property is different to ours as it is conditioned on agreement: if the Initiator and the Responder agree upon a key, then this key shall be secret, while we model secrecy of keys from both the Initiator and Responder's point of views. The minimal offensive models for the agreement properties in [DM18] are given in Table 4.2: our results broaden their findings through the inclusion of additional compromise scenarios. For the secrecy properties, the results are not directly comparable as the modeled properties are different.

Comparison with [KNB19]. This analysis also only considers static key reveals scenarios for `IKpsk2`. Translating these results to our notations, these are: a minimal offensive model for agreement on the second message is $R_s \wedge R_v$, for agreement on the transport message from the Initiator to the Responder is $R_s \wedge R_u$, for agreement on the transport message from the Responder to the Initiator is $R_s \wedge R_v$. These results address, for each property, one minimal offensive adversary model included in our security formula which encompasses all minimal offensive adversary models. For secrecy, the modeled property is different to ours as it concerns the FS of payloads (and not the secrecy of the keys). With our notations, the obtained minimal offensive adversary models are: for the FS of the payload of the second message and of the transport message from the Responder to the Initiator $R^*_s \wedge R^*_u$, for the FS of the payload of the transport message from the Initiator to the Responder $R^*_s \wedge R^*_v$.

Comparison with [GHS$^+$20]. As explained in Section 4.3, this analysis assesses all possible key compromises, and results are already security formulas (in our framework) for each property. Translating these results to our notations, the security formulas can be summarized as follows (note that [GHS$^+$20] does not distinguish which key is compromised during public key distribution, but refers

[2] `https://gitlab.limos.fr/palafour/ndss2024-AE364`

to one global compromise termed D_{pki}, furthermore, *active* refers to an active adversary):

- Agreement on the second message and on a transport message from the Responder to the Initiator: $(active \wedge R_y \wedge R_v \wedge R_s) \vee (active \wedge R_x \wedge R_u \wedge R_s) \vee (M_y \wedge R_v \wedge R_s) \vee (M_v \wedge D_{pki} \wedge R_y \wedge R_s) \vee (M_y \wedge M_v \wedge D_{pki} \wedge R_s)$.
- Agreement on the first `TransData` message from the Initiator to the Responder: $(active \wedge R_x \wedge R_u \wedge R_{psk}) \vee (active \wedge R_y \wedge R_v \wedge R_{psk}) \vee (D_x \wedge R_u \wedge R_{psk}) \vee (D_u \wedge R_x \wedge R_{psk}) \vee (D_x \wedge D_u \wedge R_{psk})$.

Hence, a minimal offensive model regarding the agreement on the first transport message from the Initiator to the Responder $D_v \wedge R_s$ which is not captured in [GHS+20] which is due to the fact that do not model the scenario of the compromise of the Responder's public key. Our model exclusively considers active adversaries, while [GHS+20] treats active behavior as an atomic capability. Consequently, certain attack scenarios achievable by passive adversaries can not be confirmed with our analysis. For secrecy, as in [KNB19], the modeled property is different to ours as it concerns secrecy and FS of payloads (and not secrecy of keys). [GHS+20] obtains the following security formulas:

- Secrecy and FS of the payload of the second message and of the transport messages from the Initiator's point of view $(R_y \wedge R_v^* \wedge R_s^*) \vee (R_x \wedge R_u^* \wedge R_s^*) \vee (M_y \wedge R_y \wedge R_s) \vee (M_v \wedge D_{pki} \wedge R_y \wedge R_s) \vee (M_y \wedge M_v \wedge D_{pki} \wedge R_s)$.
- Secrecy and FS of the payload of the second message and of the transport messages from the Responder's point of view $(R_x \wedge R_u^* \wedge R_s^*) \vee (R_y \wedge R_v^* \wedge R_s^*) \vee (D_x \wedge R_u \wedge R_s) \vee (D_u \wedge R_x \wedge R_s) \vee (D_x \wedge D_u \wedge R_s)$.

4.6 Conclusion and Discussion

In this chapter, we presented a unified symbolic analysis of WireGuard, introducing a precise Sapic+ model that consolidates and refines prior analyses. Our approach enhances the adversary model with different fine-grained atomic capabilities, including the ability to read or modify static, ephemeral, and pre-shared keys, manipulate ECDH pre-computations, and control public key distribution. This enriched model not only enables us to rediscover a known anonymity attack - previously shown only in a computational analysis - but also reveals a novel vulnerability: ECDH pre-computations can lead to concrete attacks when accessed by the attacker. To address these weaknesses, we propose countermeasures that strengthen WireGuard's security.

In addition to these results, we present a formal methodology that standardizes the presentation of the analysis results of security properties considering various adversary capabilities. This approach expresses security properties as compact formulas based on adversary capabilities. The framework enables both straightforward interpretation of results and comparison with other protocol analyses.

However, our proposed protocol model has its limitations as it is stateless. The WireGuard protocol exhibits stateful behavior through several mechanisms. First, peers must update their static public keys. Second, when overloaded and unable to respond to the Initiator, the Responder issues a cookie to be included in subsequent `InitHello` messages; persistent load conditions may prompt additional cookie transmissions requiring updates. Third, the Initiator includes a timestamp in each `InitHello` message, which the Responder stores and verifies against subsequent messages from the same Initiator - accepting only those with strictly increasing timestamps to prevent replay attacks. These stateful features - cookie updates, public key rotations, and timestamp verification - suggest that modeling WireGuard as a stateful protocol within the symbolic model represents an interesting direction for future work.

Results	**Properties**: agreement on `RecHello`, agreement on `TransData` (R to I).
[KNB19] [DM18] [GHS+20]	$(R_s \wedge R_v)$ (for `IKpsk2` with ProVerif) $(R_s \wedge R_v) \vee (R_s \wedge R_u \wedge R_x)$ (for WireGuard with Tamarin) $(D_v \wedge R_s) \vee (M_s \wedge R_v) \vee (M_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_s \wedge R_u \wedge R_x)$ (for `IKpsk2` with Tamarin)
Our work	$(D_v \wedge M_s) \vee (D_v \wedge R_s) \vee (M_s \wedge M_v) \vee (M_s \wedge R_v) \vee (M_v \wedge R_s) \vee (R_s \wedge R_v) \vee (M_i \wedge M_s \wedge M_x) \vee (M_i \wedge M_s \wedge R_x)$ $\vee(M_i \wedge M_x \wedge R_s) \vee (M_i \wedge R_s \wedge R_x) \vee (M_r \wedge M_s \wedge M_x) \vee (M_r \wedge M_s \wedge R_x) \vee (M_r \wedge M_x \wedge R_s) \vee (M_r \wedge R_s \wedge R_x)$ $\vee(M_s \wedge M_u \wedge M_x) \vee (M_s \wedge M_u \wedge R_x) \vee (M_s \wedge M_x \wedge R_c) \vee (M_s \wedge M_x \wedge R_u) \vee (M_s \wedge R_c \wedge R_x) \vee (M_s \wedge R_u \wedge R_x)$ $\vee(M_u \wedge M_x \wedge R_s) \vee (M_u \wedge R_s \wedge R_x) \vee (M_x \wedge R_c \wedge R_s) \vee (M_x \wedge R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x)$
Our work*	$(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x)$
Results	**Properties**: agreement on `TransData` (I to R).
[KNB19] [DM18] [GHS+20]	$(R_s \wedge R_u)$ (for `IKpsk2` with ProVerif) $(R_s \wedge R_u) \vee (R_s \wedge R_v \wedge R_y)$ (for WireGuard with Tamarin) $(M_s \wedge R_u) \vee (M_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_s \wedge R_v \wedge R_y)$ (for `IKpsk2` with Tamarin)
Our work	$(D_u \wedge M_s) \vee (D_u \wedge R_s) \vee (M_s \wedge M_u) \vee (M_s \wedge R_u) \vee (M_u \wedge R_s) \vee (R_s \wedge R_u) \vee (M_i \wedge M_s \wedge M_y) \vee (M_i \wedge M_s \wedge R_y)$ $\vee(M_i \wedge M_y \wedge R_s) \vee (M_i \wedge R_s \wedge R_y) \vee (M_r \wedge M_s \wedge M_y) \vee (M_r \wedge M_s \wedge R_y) \vee (M_r \wedge M_y \wedge R_s) \vee (M_r \wedge R_s \wedge R_y)$ $\vee(M_s \wedge M_v \wedge M_y) \vee (M_s \wedge M_v \wedge R_y) \vee (M_s \wedge M_y \wedge R_c) \vee (M_s \wedge M_y \wedge R_v) \vee (M_s \wedge R_c \wedge R_y) \vee (M_s \wedge R_v \wedge R_y)$ $\vee(M_v \wedge M_y \wedge R_s) \vee (M_v \wedge R_s \wedge R_y) \vee (M_y \wedge R_c \wedge R_s) \vee (M_y \wedge R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y)$
Our work*	$(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y)$
Results	**Properties**: Secrecy of k_6, C^i, C^r from Initiator's view, including PFS.
Our work	$(D_v \wedge M_s) \vee (D_v \wedge R_s) \vee (M_s \wedge M_v) \vee (M_s \wedge R_v) \vee (M_v \wedge R_s) \vee (R_s \wedge R_v) \vee (M_i \wedge M_s \wedge M_x) \vee (M_i \wedge M_s \wedge R_x)$ $\vee(M_i \wedge M_x \wedge R_s) \vee (M_i \wedge R_s \wedge R_x) \vee (M_r \wedge M_s \wedge M_x) \vee (M_r \wedge M_s \wedge R_x) \vee (M_r \wedge M_x \wedge R_s) \vee (M_r \wedge R_s \wedge R_x)$ $\vee(M_s \wedge M_u \wedge M_x) \vee (M_s \wedge M_u \wedge R_x) \vee (M_s \wedge M_x \wedge R_c) \vee (M_s \wedge M_x \wedge R_u) \vee (M_s \wedge R_c \wedge R_x) \vee (M_s \wedge R_u \wedge R_x)$ $\vee(M_u \wedge M_x \wedge R_s) \vee (M_u \wedge R_s \wedge R_x) \vee (M_x \wedge R_c \wedge R_s) \vee (M_x \wedge R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x)$ $\vee(R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$
DNF3*	$(D_v \wedge R_s) \vee (R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_x) \vee (R_s \wedge R_u \wedge R_x) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$
Results	**Properties**: Secrecy of k_6, C^i, C^r from Responder's view, including PFS.
Our work*	$(D_u \wedge M_s) \vee (D_u \wedge R_s) \vee (M_s \wedge M_u) \vee (M_s \wedge R_u) \vee (M_u \wedge R_s) \vee (R_s \wedge R_u) \vee (M_i \wedge M_s \wedge M_y) \vee (M_i \wedge M_s \wedge R_y)$ $\vee(M_i \wedge M_y \wedge R_s) \vee (M_i \wedge R_s \wedge R_y) \vee (M_r \wedge M_s \wedge M_y) \vee (M_r \wedge M_s \wedge R_y) \vee (M_r \wedge M_y \wedge R_s) \vee (M_r \wedge R_s \wedge R_y)$ $\vee(M_s \wedge M_v \wedge M_y) \vee (M_s \wedge M_v \wedge R_y) \vee (M_s \wedge M_y \wedge R_c) \vee (M_s \wedge M_y \wedge R_v) \vee (M_s \wedge R_c \wedge R_y) \vee (M_s \wedge R_v \wedge R_y)$ $\vee(M_v \wedge M_y \wedge R_s) \vee (M_v \wedge R_s \wedge R_y) \vee (M_y \wedge R_c \wedge R_s) \vee (M_y \wedge R_s \wedge R_v) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y)$ $\vee(R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$
Our work*	$(D_u \wedge R_s) \vee (R_s \wedge R_u) \vee (R_c \wedge R_s \wedge R_y) \vee (R_s \wedge R_v \wedge R_y) \vee (R_s^* \wedge R_u^* \wedge R_x) \vee (R_s^* \wedge R_v^* \wedge R_y) \vee (R_c^* \wedge R_s^* \wedge R_x \wedge R_y)$

Table 4.2: Computed security formulas for WireGuard, and comparisons with results from [KNB19], [DM18] and [GHS+20] for agreement properties. Secrecy properties are not directly comparable and anonymity is not reached for WireGuard. The simplified security formulas* consider only minimal offensive models involving atomic capabilities for secret key reveals and public key distribution compromises.

Chapter 5

Hybridization of Wireguard

Hybrid protocols aren't a stopgap - they're the only responsible transition path.

Allison Bishop

Contents

Chapter Summary

PQ-WireGuard is a post-quantum variant of WireGuard, where the Diffie-Hellman based key exchange is replaced by a post-quantum Key Encapsulation Mechanisms based key exchange. In this chapter, we first conduct a thorough formal analysis of PQ-WireGuard's original design, in which we point out and fix a number of weaknesses. This leads us to an improved construction PQ-WireGuard*. Secondly, we design and formally analyze a new protocol, based on both WireGuard and PQ-WireGuard*, named Hybrid-

WireGuard, compliant with current best practices for post-quantum transition using hybridization techniques. For our analysis, we use the Sapic$^+$ framework that enables the generation of three state-of-the-art protocol models for the verification tools ProVerif, DeepSec and Tamarin from a single specification, leveraging the strengths of each tool. Additionally, we provide a formal definition of hybrid security and formally prove that Hybrid-WireGuard satisfies this tight security notion.

5.1 Introduction

With the recent developments on the construction of quantum computers, many protocols, including VPN protocols, are transitioning to the usage of post-quantum cryptography (e.g., [Ste24, CPS19, SM16, FMJ24]) because the Diffie-Hellman key exchanges are broken by Shor's quantum factoring algorithm [Sho94], and need to be replaced. The transition is particularly urgent because of *store-now-decrypt-later* attacks, where adversaries store current data encrypted with classic primitives, and decrypt them later when efficient quantum computers emerge. In 2016, the National Institute of Standards and Technology (NIST) launched a post-quantum standardization competition for new primitives intended to replace current quantum-broken primitives. For the Diffie-Hellman key exchange, post-quantum Key Encapsulation Mechanisms (KEMs) are proposed. At the end of the third round of the competition in 2022 [AAA$^+$22], a single post-quantum KEM (Kyber [ABD$^+$19]) was selected for standardization, namely ML-KEM [MLK24]. In March 2025, NIST chose HQC [AMAD$^+$24] as the fourth-round winner for post-quantum encryption, designating it as a backup to ML-KEM. NIST plans to release a draft standard for HQC within a year [oSN25].

The security of WireGuard against quantum computing is also studied. In its original construction [Don17], a pre-shared symmetric key can be used during the handshake as a means of protection against quantum computers, instead of using post-quantum primitives which are explicitly deemed impractical for the protocol [Don24]. In 2021, a post-quantum version of WireGuard, named PQ-WireGuard, was proposed [HNS$^+$21]. It basically replaces the Diffie-Hellman shared secrets with secrets generated using post-quantum KEMs. A computational proof as well as a symbolic proof are provided in [HNS$^+$21] to ensure the security of the protocol.

Since post-quantum constructions are still new and their cryptanalysis is not fully mature, some of them could be broken in the near future, hence *genericity* (i.e., being able to replace one scheme by another in an effective way) is advised. Furthermore, during a transitional phase, *hybridization* is recommended (e.g., by the French Cybersecurity Agency (ANSSI) [ANS22]). The goal is to rely on both classic and post-quantum schemes simultaneously, ensuring that the final construction remains secure for as long as either the classic or the post-quantum scheme remains secure.

5.1.1 Our Contributions

We design and formally analyze, in the symbolic model using the SAPIC$^+$ framework, a new protocol which hybridizes WireGuard. More specifically, our contributions are the following:

- We point out a mistake in the symbolic proof of PQ-WireGuard [HNS$^+$21] using TAMARIN which has an impact on Unknown-Key-Share (UKS) attacks, and we identify attack scenarios that were missed in the previous analysis. Furthermore, PQ-WireGuard inherits the lack of anonymity from WireGuard. In addition, we point out that PQ-WireGuard uses a non-standard definition of the KEM, which may impact the protocol's implementation and analysis. The decision to employ non-standard KEM definitions was driven by the need to ensure complete resistance against Maximum Exposure (MEX) scenarios where all ephemeral keys and random values become compromised during the protocol execution, a guarantee that could not be achieved using standard definitions within their construction framework according to the authors [HNS$^+$21].

- We propose a new and more comprehensive model of PQ-WireGuard. We examine multiple attack scenarios using the combination of the adversary's capabilities and rigorously verify an extensive set of security properties. For each property, we derive a precise security formula that characterizes the exact conditions for its violation, accounting for the attacker's capabilities, as defined in Chapter 4 framework.

- We therefore propose a new version of the protocol, based on the common KEM's definition, which we refer to as PQ-WireGuard*. We prove that this enhanced version guarantees anonymity and resistance to UKS attacks while maintaining all other security properties (agreement and secrecy) even against a stronger adversary than considered in the original protocol [HNS$^+$21]. While following the standard KEM definitions, we demonstrate protocol resistance to MEX attacks under complete compromise of the protocol's randomness.

- We propose a new protocol, called *Hybrid-WireGuard*, constructed by combining our improved construction PQ-WireGuard*, and the original WireGuard. We model this new protocol in the applied Π-CALCULUS and prove that all the verified security properties are achieved. Moreover, we provide a formal definition of hybrid security and prove that Hybrid-WireGuard satisfies this security notion. Our analysis shows that if an attack scenario against a security property of Hybrid-WireGuard exists, then it is a combination of an attack on PQ-WireGuard* and an attack on the original WireGuard. Essentially, to compromise Hybrid-WireGuard, an adversary must successfully attack both the post-quantum and classical components of the protocol. This dual-security requirement is the fundamental principle behind hybridization.

5.1.2 Related Work

According to the French Cybersecurity Agency, emerging post-quantum KEMs and signatures are not mature enough to be confidently used on their own in security products [ANS22]. Hence, hybridization is currently advised during a transitional period [ANS22, Nat23, Fed21], where post-quantum schemes are simultaneously used with classic ones. Hybridization is not trivial as combining classic and post-quantum primitives must be done without introducing further security flaws. Also, the final construction's security should be ensured as long as at least the security of the classic or the security of the post-quantum primitive is ensured. The European Telecommunications Standards Institute (ETSI), for example, discusses techniques for hybrid key exchange [Eur20]. Many protocols and implementations already include hybrid solutions in their constructions. OpenSSH, for instance, recently implemented in version 9.9 an hybrid key exchange using X25519 and ML-KEM [KSH24]. In addition, numerous works provide performance analysis on hybrid variants of the TLS handshake [CPS19, PST20, SKD20, SM16]. In [SSW20], the authors introduce KEMTLS, a variant of TLS using KEMs instead of signatures for authentication, formally verified in [CHSW22] with Tamarin. In the secure messaging domain, the Signal messaging protocol introduces a hybrid key agreement protocol [KS23]. Apple's IMessage also introduces PQ3 [Ste24], a secure hybrid messaging protocol. A computational proof [Ste24] and a symbolic proof of the latter protocol using Tamarin [LSB24] are also available. For VPNs, a hybrid key exchange [TTB+23] is proposed for the IKEv2 protocol [KHNE10], later implemented by StrongSwan6.0 [pqs], using the liboqs post-quantum library [SM16]. Our construction of Hybrid-WireGuard follows this line of research.

Towards a Post-Quantum WireGuard. Beyond [HNS+21], other proposals have also explored post-quantum adaptations of WireGuard. In [AMW19], a tweak is proposed to provide WireGuard with post-quantum security. It consists in transmitting the hash of the public key instead of the public key itself, hence protecting it from a quantum computer. The goal is to protect against *store-now-decrypt-later* attacks. However, this tweak relies on the strong assumption that the peers' public keys are unknown to the attacker (a similar assumption is done in the original design of WireGuard [Don17]). NordLynx [Nor], a variant of WireGuard, proposes a post-quantum solution: after the successful handshake, a KEM key exchange is performed using ML-KEM, within the already mounted WireGuard tunnel. This solution provides post-quantum security for the final session keys, mitigating attacks from adversaries who can only observe the exchanged messages (i.e., *passive* adversaries). Rosenpass [VLZ+24] is an implementation of PQ-WireGuard [HNS+21], which aims to provide peers with a post-quantum shared secret. This secret can then be used by any protocol handshake to add a layer of post-quantum security to the session keys. Typically, a VPN tunnel can be mounted by sequentially executing Rosenpass to generate a post-quantum pre-shared key, and then executing a classic VPN protocol, such as WireGuard, with the pre-shared key as input. In this work, we use a single 2-message key exchange to provide hybrid (i.e., classic and post-quantum) security, by simultaneously using ECDH and KEM key exchanges.

Symbolic Analysis of KEM-based Protocols. Many symbolic models for KEM-based protocols have been proposed in the literature [BJKS24, CHSW22, HNS+21, CDM24]. Most works (except for that of [CDM24]) rely on equational theories, a common approach for modeling cryptographic primitives in the symbolic model, and typically use the same theory as for asymmetric encryption. In contrast, Cremers et al. [CDM24] employ *restrictions* to define KEM properties and develop a TAMARIN library for analyzing security protocols using KEMs. While our approach models KEMs only via asymmetric encryption equations, we nevertheless identified a re-encapsulation attack, described in Section 5.3.3.

Outline. We begin by giving a full description of the PQ-WireGuard handshake and compare it to the handshake of WireGuard in Section 5.2. In Section 5.3, we describe our methodology for the formal verification of our protocols and present the verified security properties. We exhibit the results of our symbolic analysis for PQ-WireGuard in Section 5.3.3. Our results on PQ-WireGuard lead us to an improved version of the protocol, which we refer to as PQ-WireGuard*, described and symbolically analyzed in the same section. Later, we introduce our new protocol, namely Hybrid-WireGuard, in Section 5.4. We give a formal definition of hybrid security and present the results of our symbolic analysis of the Hybrid-WireGuard in the same section. Finally, we present our conclusions and outline directions for future work in Section 5.5.

5.2 PQ-WireGuard

We describe the PQ-WireGuard [HNS+21] handshake and compare it to the handshake of WireGuard given in Section 4.2 of Chapter 4. We use alternative notations to Chapter 4's to streamline the presentation of our results. We start by recalling the cryptographic building blocks used by both constructions.

5.2.1 Cryptographic Building Blocks

As established in Chapter 4, WireGuard uses the Diffie-Hellman (ECDH) key exchange, based on a cyclic group $\mathbb{G}$ of generator g. We use $\mathsf{DH.gen}()$ to generate the ECDH keypair (s, S) where s is generated uniformly at random and $\mathsf{S} = g^{\mathsf{s}}$. We also use an ECDH computation that takes as input the private key of one ECDH key pair, and the public key of another, i.e., either (s_1, S_2) or (S_1, s_2), to generate the ECDH shared secret. $\mathsf{KEM.gen}()$, $\mathsf{KEM.encaps}(\mathsf{S})$, $\mathsf{KEM.decaps}(\mathsf{s}, ct)$ denote the keypair generation, the probabilistic encapsulation algorithm, and the decapsulation algorithm respectively as described in Section 2.3.4 of Chapter 2. $\mathsf{AEAD.Enc}(k, N, H, M)$ and $\mathsf{AEAD.Dec}(k, N, H, C)$ denote respectively the symmetric authenticated encryption and decryption as introduced in Section 2.3.3 of Chapter 2. Hash denotes a cryptographic hash function, MAC denotes the message authenticating code, and KDF_n is a key derivation function (with key material k and some input I) indexed with the integer n. We denote the ℓ-th output by $\mathsf{KDF}_n(k, I)[\ell]$.

5.2.2 The Handshake

To ease comparison between both constructions, all algorithms and key derivations are simplified and presented in Tables 5.2 on page 116, 5.3 on page 117 and 5.14 on page 139, with notations from Table 5.1.

WireGuard handshake. We give a high level description of WireGuard's handshake in Table 5.2 on page 116 as a reminder from Chapter 4. The handshake consists of two messages: `InitHello` sent by the Initiator, followed by `RespHello` sent by the Responder. Both the Initiator and the Responder hold long-term ECDH keys $(\mathsf{s}_\mathsf{i}^\mathsf{c}, \mathsf{S}_\mathsf{i}^\mathsf{c})$ and $(\mathsf{s}_\mathsf{r}^\mathsf{c}, \mathsf{S}_\mathsf{r}^\mathsf{c})$, respectively. Table 5.14 on page 139 describes the key derivation chain done by both peers to agree on the final session keys: tk_i used to encrypt traffic sent by the Initiator, and tk_r used to encrypt traffic sent by the Responder. Both the Initiator and the Responder generate ephemeral ECDH keys $(\mathsf{e}_\mathsf{i}^\mathsf{c}, \mathsf{E}_\mathsf{i}^\mathsf{c})$ and $(\mathsf{e}_\mathsf{r}^\mathsf{c}, \mathsf{E}_\mathsf{r}^\mathsf{c})$ respectively during the handshake. Then, a total of four ECDH shared secrets are used in the key derivation. These four secrets correspond to the different combinations of static and ephemeral ECDH keys of both peers: $\mathsf{dh}_{\mathsf{s_i s_r}}$, $\mathsf{dh}_{\mathsf{s_i e_r}}$, $\mathsf{dh}_{\mathsf{e_i s_r}}$ and $\mathsf{dh}_{\mathsf{e_i e_r}}$.

ECDH **keys**	Static	Ephemeral
Initiator	(s_i^c, S_i^c)	(e_i^c, E_i^c)
Responder	(s_r^c, S_r^c)	(e_r^c, E_r^c)

ECDH **shared secrets**		Responder	
		Static	Ephemeral
Initiator	Static	$dh_{s_i s_r}$	$dh_{s_i e_r}$
	Ephemeral	$dh_{e_i s_r}$	$dh_{e_i e_r}$

KEM **keys**	Static	Ephemeral
Initiator	(s_i^{pq}, S_i^{pq})	(e_i^{pq}, E_i^{pq})
Responder	(s_r^{pq}, S_r^{pq})	-

KEM.encaps **outputs**	Secret key	Ciphertext
S_i^{pq}	shk_3	ct_3
E_i^{pq}	shk_2	ct_2
S_r^{pq}	shk_1	ct_1

KEM.encaps **randoms**	Static	Ephemeral
Initiator	σ_i	r_i
Responder	σ_r	r_r, r_e

Table 5.1: Notations for ECDH keys and shared secrets, KEM keys, encapsulation outputs and andom encapsulation inputs. Key pairs are denoted as (private key, public key). $*^c$ denotes a classic key, $*^{pq}$ a post-quantum key. The random inputs are specific to PQ-WireGuard [HNS+21].

PQ-WireGuard handshake. Since the ECDH key exchange is not post quantum secure, it needs to be replaced by post-quantum KEMs. In [HNS+21], the authors replace the ECDH shared secrets by ones generated using KEMs such that the same security properties are ensured. In this case, both the Initiator and the Responder hold static keys (s_i^{pq}, S_i^{pq}) and (s_r^{pq}, S_r^{pq}), that correspond to a post-quantum KEM. We describe the handshake and the key derivation, as proposed in [HNS+21], in Tables 5.3 on page 117 and 5.14 on page 139 respectively. The differences from WireGuard are highlighted in the tables.

ECDH shared secrets that involve at least one ephemeral key are easily replaced by KEMs. The one corresponding to $dh_{s_i e_r}$ is generated by the Responder using S_i^{pq} (shk_3 on line 4 for `RespHello`). The one corresponding to $dh_{e_i s_r}$ is generated by the Initiator using S_r^{pq} (shk_1 on line 4 for `InitHello`). The one corresponding to $dh_{e_i e_r}$ is generated by the Responder using the the Initiator's ephemeral key E_i^{pq} (shk_2 on line 3 for `RespHello`). Meanwhile, shared secret $dh_{s_i s_r}$, generated

1. `InitHello` construction

```
input: $s_i^c, S_i^c, S_r^c$
$\mathsf{sid}_i \xleftarrow{\$} \{0,1\}^{32}$
$(e_i^c, E_i^c) \leftarrow \mathsf{DH.gen}()$
$\mathsf{static} \leftarrow \mathsf{AEAD.Enc}(\kappa_3,\ 0,\ H_3,\ S_i^c)$
$\mathsf{time} \leftarrow \mathsf{AEAD.Enc}(\kappa_4,\ 0,\ H_4,\ \mathsf{now}())$
$\mathsf{inner} \leftarrow \mathsf{type} \mid\mid \mathsf{sid}_i \mid\mid E_i^c \mid\mid \mathsf{static} \mid\mid \mathsf{time}$
$\mathsf{m1} \leftarrow \mathsf{MAC}(\mathsf{Hash}(\mathsf{lbl3} \mid\mid S_r^c),\ \mathsf{inner})$
$\mathsf{m2} \leftarrow \mathsf{MAC}(\mathsf{cookie},\ \mathsf{inner} \mid\mid \mathsf{m1})$
$\mathsf{InitHello} \leftarrow \mathsf{inner} \mid\mid \mathsf{m1} \mid\mid \mathsf{m2}$
```

2. `RespHello` construction

```
input: $s_r^c, S_r^c, S_i^c$
$\mathsf{sid}_r \xleftarrow{\$} \{0,1\}^{32}$
$(e_r^c, E_r^c) \leftarrow \mathsf{DH.gen}()$
$\mathsf{empty} \leftarrow \mathsf{AEAD.Enc}(\kappa_9,\ 0,\ H_9,\ \varnothing)$

$\mathsf{inner} \leftarrow \mathsf{type} \mid\mid \mathsf{sid}_r \mid\mid \mathsf{sid}_i \mid\mid E_r^c \mid\mid \mathsf{empty}$
$\mathsf{m1} \leftarrow \mathsf{MAC}(\mathsf{Hash}(\mathsf{lbl3} \mid\mid S_i^c),\ \mathsf{inner})$
$\mathsf{m2} \leftarrow \mathsf{MAC}(\mathsf{cookie},\ \mathsf{inner} \mid\mid \mathsf{m1})$
$\mathsf{RespHello} \leftarrow \mathsf{inner} \mid\mid \mathsf{m1} \mid\mid \mathsf{m2}$
```

Table 5.2: WireGuard [Don17] handshake messages.

using the static keys of the peers, cannot be replaced by a secret generated using a KEM key exchange because the latter requires at least one interaction. Observe that $\mathsf{dh}_{\mathsf{s_i s_r}}$ binds both identities of the peers to the key derivation. It also ensures that the first sent message `InitHello` is already authenticated, helping to mitigate Denial-of-Service (DoS) attacks. To replace this secret, the authors of [HNS+21] impose using a pre-shared key psk (optional in WireGuard). They discuss that one may rely on the assumption that public keys are actually not public and hence unknown to attackers, similarly to other works using this assumption [Don17, AMW19], making the use of the value $\mathsf{psk} \leftarrow \mathsf{Hash}(S_i^{pq} \oplus S_r^{pq})$ enough for security.

In addition, the shared secret $\mathsf{dh}_{\mathsf{s_i s_r}}$ does not rely on any ephemeral randomness, adding a layer of protection against potential random state corruption. For this reason, the authors redefine the KEM.encaps procedure to make it deterministic. The random coins are instead provided as input, so that the shared secrets are derived from a trusted source of randomness. To generate the secret corresponding to $\mathsf{dh}_{\mathsf{e_i s_r}}$, the Initiator combines an ephemeral random $\mathsf{r_i}$ with a static one using KDF_1 and provides the output as input to KEM.encaps. The Responder does the same during `RespHello`'s construction to generate the secret corresponding to $\mathsf{dh}_{\mathsf{s_i e_r}}$. Finally, the secret corresponding to $\mathsf{dh}_{\mathsf{e_i e_r}}$, involves only ephemeral coins. We give more details about the security properties achieved by PQ-WireGuard in Section 5.3. Finally, for their computational proof in [HNS+21], the authors choose the IND-CCA [BDPR98] security assumption for the KEM corresponding to the static keys, and IND-CPA [BDPR98] security assumption for the KEM corresponding to the ephemeral keys.

1. `InitHello` construction

```
input: $\sigma_i, s_i^{pq}, S_i^{pq}, S_r^{pq}$
$\mathsf{sid}_i \xleftarrow{\$} \{0,1\}^{32}$
$\mathsf{r_i} \xleftarrow{\$} \{0,1\}^{256}$
$(\mathsf{e}_i^{pq}, \mathsf{E}_i^{pq}) \leftarrow \mathsf{KEM.gen}()$
$(\mathsf{ct}_1, \mathsf{shk}_1) \leftarrow \mathsf{KEM.Encaps}(\mathsf{S}_r^{pq},\ KDF_1(\sigma_i, \mathsf{r_i}))$
$\mathsf{static} \leftarrow \mathsf{AEAD.Enc}(\kappa_3,\ 0,\ H_3,\ \mathsf{Hash}(\mathsf{S}_i^{pq}))$
$\mathsf{time} \leftarrow \mathsf{AEAD.Enc}(\kappa_4,\ 0,\ H_4,\ \mathsf{now}())$
$\mathsf{inner} \leftarrow \mathsf{type} \mid\mid \mathsf{sid}_i \mid\mid \mathsf{E}_i^{pq} \mid\mid \mathsf{ct}_1 \mid\mid \mathsf{static} \mid\mid \mathsf{time}$
$\mathsf{m1} \leftarrow \mathsf{MAC}(\mathsf{Hash}(\mathsf{lbl3} \mid\mid \mathsf{S}_r^{pq}),\ \mathsf{inner})$
$\mathsf{m2} \leftarrow \mathsf{MAC}(\mathsf{cookie},\ \mathsf{inner} \mid\mid \mathsf{m1})$
$\mathsf{InitHello} \leftarrow \mathsf{inner} \mid\mid \mathsf{m1} \mid\mid \mathsf{m2}$
```

2. `RespHello` construction

```
input: $\sigma_r, s_r^{pq}, S_r^{pq}, S_i^{pq}$
$\mathsf{sid}_r \xleftarrow{\$} \{0,1\}^{32}$
$\mathsf{r_e},\ \mathsf{r_r} \xleftarrow{\$} \{0,1\}^{256} \times \{0,1\}^{256}$
$(\mathsf{ct}_2, \mathsf{shk}_2) \leftarrow \mathsf{KEM.Encaps}(\mathsf{E}_i^{pq},\ \mathsf{r_e})$
$(\mathsf{ct}_3, \mathsf{shk}_3) \leftarrow \mathsf{KEM.Encaps}(\mathsf{S}_i^{pq},\ KDF_1(\sigma_r, \mathsf{r_r}))$
$\mathsf{empty} \leftarrow \mathsf{AEAD.Enc}(\kappa_9,\ 0,\ H_9,\ \varnothing)$

$\mathsf{inner} \leftarrow \mathsf{type} \mid\mid \mathsf{sid}_r \mid\mid \mathsf{sid}_i \mid\mid \mathsf{ct}_2 \mid\mid \mathsf{ct}_3 \mid\mid \mathsf{empty}$
$\mathsf{m1} \leftarrow \mathsf{MAC}(\mathsf{Hash}(\mathsf{lbl3} \mid\mid \mathsf{S}_i^{pq}),\ \mathsf{inner})$
$\mathsf{m2} \leftarrow \mathsf{MAC}(\mathsf{cookie},\ \mathsf{inner} \mid\mid \mathsf{m1})$
$\mathsf{RespHello} \leftarrow \mathsf{inner} \mid\mid \mathsf{m1} \mid\mid \mathsf{m2}$
```

Table 5.3: PQ-WireGuard [HNS+21] handshake messages.

5.3 Formal Analysis and Claimed Properties

Our analysis follows the methodological framework established in Sections 4.4 and 4.5 of Chapter 4 to analyze PQ-WireGuard. We first define the set of atomic capabilities to be considered. Then, for each security property, we derive the corresponding security formula (per Definition 4.4.5) using ProVerif. Unlike in the previous chapter, we do not use Tamarin here because our equational theory does not involve Associative and Commutative (AC) function symbols. For trace properties, we simply rely on ProVerif. For equivalence properties, we additionally confirm the formulas found using ProVerif with DeepSec.

5.3.1 Protocol Model

We model the protocol as processes, as is standard in the applied Π-Calculus, adhering as closely as possible to the protocol specifications in [HNS+21]. Specific events are inserted into each process to enable reasoning about security properties. The events in question are described in Table 5.4.

ISend$(\mathsf{ck}_1, \mathsf{S}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^\mathsf{pq}, \sigma_i, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}, \mathsf{shk}_1)$:
the event inserted into the initiator's process with public key $\mathsf{S}_\mathsf{i}^\mathsf{pq}$ at the point where they send the `InitHello` message to the peer with public key $\mathsf{S}_\mathsf{r}^\mathsf{pq}$ encrypted with the key ck_1.
RRec$(\mathsf{ck}_1, \mathsf{S}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^\mathsf{pq}, \sigma_r, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}, \mathsf{shk}_1)$:
the event inserted into the responder's process with public key $\mathsf{S}_\mathsf{r}^\mathsf{pq}$ at the point where they receive and accept an `InitHello` message from the peer with public key $\mathsf{S}_\mathsf{i}^\mathsf{pq}$ encrypted with the key ck_1.
RKeys$(\mathsf{ck}_2, \mathsf{S}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^\mathsf{pq}, \sigma_r, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}, \mathsf{shk}_1, \mathsf{shk}_3, \mathsf{shk}_2)$:
the event inserted into the responder's process with public key $\mathsf{S}_\mathsf{r}^\mathsf{pq}$ at the point where they send a `RecHello` message to the peer with public key $\mathsf{S}_\mathsf{i}^\mathsf{pq}$ encrypted with the key ck_2.
IKeys$(\mathsf{ck}_2, \mathsf{S}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^\mathsf{pq}, \sigma_i, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}, \mathsf{shk}_1, \mathsf{shk}_3, \mathsf{shk}_2)$:
the event inserted into the initiator's process with public key $\mathsf{S}_\mathsf{i}^\mathsf{pq}$ at the point where they receive and accept a `RecHello` message from the peer with public key $\mathsf{S}_\mathsf{r}^\mathsf{pq}$ encrypted with the key ck_2.
IConfirm$(\mathsf{ck}_3, \mathsf{S}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^\mathsf{pq}, \sigma_i, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}, \mathsf{shk}_1, \mathsf{shk}_3, \mathsf{shk}_2)$:
the event inserted into the initiator's process with public key $\mathsf{S}_\mathsf{i}^\mathsf{pq}$ at the point where they confirm the handshake with the peer with public key $\mathsf{S}_\mathsf{i}^\mathsf{pq}$ encrypted with the session key ck_3.
RConfirm$(\mathsf{ck}_3, \mathsf{S}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^\mathsf{pq}, \sigma_r, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}, \mathsf{shk}_1, \mathsf{shk}_3, \mathsf{shk}_2)$:
the event inserted into the responder's process with public key $\mathsf{S}_\mathsf{r}^\mathsf{pq}$ at the point where they confirm the handshake with the peer with public key $\mathsf{S}_\mathsf{r}^\mathsf{pq}$ encrypted with the session key ck_3.

Table 5.4: Specification of the inserted events.

5.3.2 Claimed Security Properties

The authors of PQ-WireGuard argue that their protocol preserves several security properties, which they expect to carry over from the established security guarantees of classical WireGuard. In addition, PQ-WireGuard should be resilient to many attack scenarios involving key compromises. In this section, we systematically examine each of these claimed properties, providing precise formal definitions alongside their intuitive interpretations to enable comprehensive analysis and comparison. Table 5.5 compares the list of claimed security properties, and the properties we evaluated with those from previous symbolic analyses for WireGuard [DM17] and PQ-WireGuard [HNS$^+$21].

Property	[DM17]	[HNS$^+$21]	**Our work**
Maximal Exposure (MEX) attacks resistance	✗	✓^{t}	✓^{t,e}
Unilateral Unknown Key Share (UUKS) attacks resistance	✗	✓^{t}	✓^{t}
Bilateral Unkown Key Share (BUKS) attacks resistance	✗	✗	✓^{t}
Session uniqueness	✓^{t}	✓^{t}	✓^{t}
Anonymity	✓^{t}	✓^{t}	✓^{e}
Message agreement	✓^{t}	✓^{t}	✓^{t}
Key Compromise Impersonation (KCI) attacks resistance	✓^{t}	✓^{t}	✓^{t}
Key secrecy	✓^{t}	✓^{t}	✓^{t}
Strong key secrecy	✗	✗	✓^{e}
Mutual key secrecy	✗	✓^{t}	✓^{t}
Key forward secrecy	✓^{t}	✓^{t}	✓^{t}

✓: **analyzed** ✗: **not analyzed**

Table 5.5: Security properties, analyzed as trace (t) or equivalence (e) properties. [DM17] models Wireguard and [HNS$^+$21] models PQ-WireGuard.

Resistance against Maximal Exposure (MEX) attacks. The authors of [Kra05, FSXY12] state that a MEX attack occurs when an adversary successfully distinguishes the session key from a random value, given access to any pair of static and ephemeral secret keys except for both the static and ephemeral secret keys of the initiator or the responder. When analyzing PQ-Wireguard, the authors of [HNS$^+$21] extended the latter security definition in two ways. First, they made it applicable to any type of security property, not just session key secrecy. Second, they allowed for any possible combination of static and ephemeral key compromises. However, for each security property they examined, they excluded specific corruption cases where attacks are unavoidable. By expressing security properties as formal security formulas (following Definition 4.4.5), we explicitly capture the precise conditions of secret key compromises under which each security property holds or fails. This provides a direct way for analyzing resistance against MEX attacks, as these scenarios are directly encoded within the formulas themselves. Thus, we obtain these results directly from our formulas at no additional cost.

Resistance against Key Compromise Impersonation (KCI) attacks. KCI attacks occur when an attacker gains access to the static secret keys material of a peer, and is able to impersonate their corresponding honest peer during a run of the protocol. We do not model this property explicitly as in [HNS+21], instead it is directly deduced from our security formulas since the scenarios where the attacker gains access to the static key materials of peers is covered by our attacker's atomic capabilities.

Resistance against Unknown-Key-Share (UKS) attacks. UKS attacks enable an attacker to coerce honest peers into exchanging keys with parties other than the ones they believe they are communicating with, without being aware of this exchange. In [CT07], Liqun et al. define Unilateral UKS (UUKS) attacks as scenarios where either the initiator is coerced to accept an incorrect responder identity or the responder is coerced to accept an incorrect initiator identity. They define Bilateral UKS (BUKS) attacks as scenarios where both parties are coerced to establish a shared key while each believes they are communicating with a different entity. We analyzed those properties for the Initiator and the Responder (unilateral case) and for both peers (bilateral case), and they are formalized in Table 5.6.

	$\forall\ \mathsf{S}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^{\mathsf{pq}\prime}, \mathsf{S}_\mathsf{i}^{\mathsf{pq}\prime}, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}, \mathsf{psk}', \mathrm{ck}, \mathsf{shk}_1, \mathsf{shk}_3, \mathsf{shk}_2, \sigma_i, \sigma_r;$
Init. UUKS.	$\big(\mathbf{event}(\mathbf{RConfirm}(\mathrm{ck}, \mathsf{S}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^\mathsf{pq}, \sigma_r, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}, \mathsf{shk}_1, \mathsf{shk}_3, \mathsf{shk}_2))\wedge$ $\mathbf{event}(\mathbf{IConfirm}(\mathrm{ck}, \mathsf{S}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^{\mathsf{pq}\prime}, \sigma_i, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}', \mathsf{shk}_1, \mathsf{shk}_3, \mathsf{shk}_2))\big)$ $\Rightarrow \big((\mathsf{S}_\mathsf{r}^\mathsf{pq} = \mathsf{S}_\mathsf{r}^{\mathsf{pq}\prime}) \wedge (\mathsf{psk} = \mathsf{psk}')\big).$
Resp. UUKS.	$\big(\mathbf{event}(\mathbf{RConfirm}(\mathrm{ck}, \mathsf{S}_\mathsf{i}^{\mathsf{pq}\prime}, \mathsf{S}_\mathsf{r}^\mathsf{pq}, \sigma_r, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}', \mathsf{shk}_1, \mathsf{shk}_3, \mathsf{shk}_2))\wedge$ $\mathbf{event}(\mathbf{IConfirm}(\mathrm{ck}, \mathsf{S}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^\mathsf{pq}, \sigma_i, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}, \mathsf{shk}_1, \mathsf{shk}_3, \mathsf{shk}_2))\big)$ $\Rightarrow \big((\mathsf{S}_\mathsf{i}^\mathsf{pq} = \mathsf{S}_\mathsf{i}^{\mathsf{pq}\prime}) \wedge (\mathsf{psk} = \mathsf{psk}')\big).$
BUKS	$\big(\mathbf{event}(\mathbf{RConfirm}(\mathrm{ck}, \mathsf{S}_\mathsf{i}^{\mathsf{pq}\prime}, \mathsf{S}_\mathsf{r}^\mathsf{pq}, \sigma_r, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}, \mathsf{shk}_1, \mathsf{shk}_3, \mathsf{shk}_2))\wedge$ $\mathbf{event}(\mathbf{IConfirm}(\mathrm{ck}, \mathsf{S}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^{\mathsf{pq}\prime}, \sigma_i, \mathsf{E}_\mathsf{i}^\mathsf{pq}, \mathsf{psk}', \mathsf{shk}_1, \mathsf{shk}_3, \mathsf{shk}_2))\big)$ $\Rightarrow \big((\mathsf{S}_\mathsf{i}^\mathsf{pq} = \mathsf{S}_\mathsf{i}^{\mathsf{pq}\prime}) \wedge (\mathsf{S}_\mathsf{r}^\mathsf{pq} = \mathsf{S}_\mathsf{r}^{\mathsf{pq}\prime}) \wedge (\mathsf{psk} = \mathsf{psk}')\big).$

Table 5.6: Resistance to Unknown-Key-Share attack properties.

Session uniqueness. This property states that a session key computed on each side is unique for each session. We analyze session uniqueness as in [HNS+21] on the Initiator and the Responder sides, and it is formalized in Table 5.7.

Message agreement. This property corresponds to full agreement, as defined in [Low97]: if a peer A receives a message, apparently from another peer B, then B has previously been running the protocol with A, and both peers agreed

	$\forall\ S_i^{pq}, S_r^{pq}, E_i^{pq}, E_i^{pq\prime}, psk, ck, shk_1, shk_3, shk_2, shk_3{}', shk_2{}', \sigma_i, \sigma_r;$
Init. Side	$\big(\mathbf{event}(\mathbf{IConfirm}(ck, S_i^{pq}, S_r^{pq}, \sigma_i, E_i^{pq}, psk, shk_1, shk_3, shk_2)) \wedge$ $\mathbf{event}(\mathbf{IConfirm}(ck, S_i^{pq}, S_r^{pq}, \sigma_i, E_i^{pq\prime}, psk, shk_1{}', shk_3, shk_2))\big)$ $\Rightarrow \big((E_i^{pq} = E_i^{pq\prime}) \wedge (shk_1{}' = shk_1)\big).$
Resp. Side	$\big(\mathbf{event}(\mathbf{RConfirm}(ck, S_i^{pq}, S_r^{pq}, \sigma_r, E_i^{pq}, psk, shk_1, shk_3, shk_2)) \wedge$ $\mathbf{event}(\mathbf{RConfirm}(ck, S_i^{pq}, S_r^{pq}, \sigma_r, E_i^{pq\prime}, psk, shk_1, shk_3{}', shk_2{}'))\big)$ $\Rightarrow \big((E_i^{pq} = E_i^{pq\prime}) \wedge (shk_3 = shk_3{}') \wedge (shk_2 = shk_2{}')\big).$

Table 5.7: Session uniqueness properties.

on sent, received and atomic data used in the protocol. We consider agreement on the `InitHello`, `RecHello`, and `Confirm` messages. We note that agreement on `RecHello` message enables the Initiator to authenticate the Responder, and agreement on the `Confirm` message (which we also refer to as the first `TransData` message) enables the Responder to authenticate the Initiator. When both properties are verified, then mutual authentication holds. We examined the agreement on the first message `InitHello` to assess whether the Responder can authenticate the Initiator based only on this initial message. The properties are formalized in Table 5.8.

	$\forall\ S_i^{pq}, S_r^{pq}, E_i^{pq}, psk, ck, shk_1, shk_3, shk_2, \sigma_i, \sigma_r;$
`InitHello`	$\mathbf{event}(\mathbf{RRec}(ck, S_i^{pq}, S_r^{pq}, \sigma_i, E_i^{pq}, psk, shk_1)) \Rightarrow$ $\mathbf{event}(\mathbf{ISend}(ck, S_i^{pq}, S_r^{pq}, E_i^{pq}, psk, shk_1)).$
`RespHello`	$\mathbf{event}(\mathbf{IKeys}(ck, S_i^{pq}, S_r^{pq}, \sigma_i, E_i^{pq}, psk, shk_1, shk_3, shk_2)) \Rightarrow$ $\mathbf{event}(\mathbf{RKeys}(ck, S_i^{pq}, S_r^{pq}, \sigma_r, E_i^{pq}, psk, shk_1, shk_3, shk_2)).$
Confirm	$\mathbf{event}(\mathbf{RConfirm}(ck, S_i^{pq}, S_r^{pq}, \sigma_r, E_i^{pq}, psk, shk_1, shk_3, shk_2)) \Rightarrow$ $\mathbf{event}(\mathbf{IConfirm}(ck, S_i^{pq}, S_r^{pq}, \sigma_i, E_i^{pq}, psk, shk_1, shk_3, shk_2)).$

Table 5.8: Agreement properties.

Key secrecy. The session key should remain secret to the attacker. We consider secrecy as a trace property, and we analyze secrecy for the session key from three perspectives: the Initiator's point of view, the Responder's point of view (since peers may not agree on the same keys), and mutual forward secrecy (as done in [HNS$^+$21]), which combines key secrecy and agreement on the same key. Secrecy properties are formulized in Table 5.9. We note that **attacker** is a ProVerif predicate used in the queries to reason about the attacker's knowledge, i.e., **attacker**(x) means that the attacker knows x.

	$\forall\ \mathsf{S_i^{pq}}, \mathsf{S_r^{pq}}, \mathsf{E_i^{pq}}, \mathsf{psk}, \mathrm{ck}, \mathsf{shk_1}, \mathsf{shk_3}, \mathsf{shk_2}, \sigma_i, \sigma_r;$
Init.'s point of view	$\mathbf{event}(\mathbf{IConfirm}(\mathrm{ck}, \mathsf{S_i^{pq}}, \mathsf{S_r^{pq}}, \sigma_i, \mathsf{E_i^{pq}}, \mathsf{psk}, \mathsf{shk_1}, \mathsf{shk_3}, \mathsf{shk_2})) \wedge$ $\mathbf{attacker}(\mathrm{ck}).$
Resp.'s point of view	$\mathbf{event}(\mathbf{RConfirm}(\mathrm{ck}, \mathsf{S_i^{pq}}, \mathsf{S_r^{pq}}, \sigma_r, \mathsf{E_i^{pq}}, \mathsf{psk}, \mathsf{shk_1}, \mathsf{shk_3}, \mathsf{shk_2})) \wedge$ $\mathbf{attacker}(\mathrm{ck}).$
Mutual secrecy	$\mathbf{event}(\mathbf{RConfirm}(\mathrm{ck}, \mathsf{S_i^{pq}}, \mathsf{S_r^{pq}}, \sigma_r, \mathsf{E_i^{pq}}, \mathsf{psk}, \mathsf{shk_1}, \mathsf{shk_3}, \mathsf{shk_2})) \wedge$ $\mathbf{event}(\mathbf{IConfirm}(\mathrm{ck}, \mathsf{S_i^{pq}}, \mathsf{S_r^{pq}}, \sigma_i, \mathsf{E_i^{pq}}, \mathsf{psk}, \mathsf{shk_1}, \mathsf{shk_3}, \mathsf{shk_2})) \wedge$ $\mathbf{attacker}(\mathrm{ck}).$

Table 5.9: Secrecy properties.

Key forward secrecy. Key forward secrecy for key exchange protocols is verified "if the compromise of long-term secrets does not lead to the compromise of session keys of previously completed sessions" [BG19]. For PQ-WireGuard, this translates to ensuring that the compromise of the pre-shared key psk, the secret static keys $\mathsf{s_i^{pq}}$ and $\mathsf{s_r^{pq}}$, and the long-term randomness σ_i and σ_r, does not lead to the compromise of session keys from previously established sessions. We analyze forward secrecy from the Initiator's view, the Responder's view, and mutual forward secrecy, following the same approach as for secrecy earlier. In ProVerif, these properties are "formalized" as in Table 5.9, with the difference that, in the model, all static keys are revealed to the attacker in a later phase, i.e., after the protocol execution, using the predicate **phase**.

Strong key secrecy. In [HNS+21], the authors define session keys secrey as keys being "*indistinguishable from a random string*" to the attacker. At the same time, in their symbolic analysis, they only consider a weaker definition of secrecy, expressed as a trace property, i.e., the attacker cannot recover the entire secret. Analyzing whether a term is indistinguishable from a freshly generated random value requires defining secrecy as an equivalence property, which makes this property stronger. We examine both definitions: we employ the term strong key secrecy to refer to the equivalence property as presented in Table 5.5. We model strong key secrecy as an equivalence property between a protocol execution in which the session key is computed as specified by the protocol, and the protocol execution in which the session key is replaced by a freshly generated name. This property is refered to as "*Key Indistinguishability*" in [BCW22].

Anonymity. Anonymity, refered to as *Identity Hiding* in [HNS+21], ensures that a user is able to participate in the protocol without an attacker being able to draw conclusions about their identity. While identity hiding was formalized as a trace property, we maintain that this is insufficient and only represents a weak property that can be viewed as "secrecy of identities", given that public keys are incorporated into the messages. We follow the common practice in symbolic analysis, that is, expressing privacy properties as equivalence properties [BCW22]. Anonymity is defined as an equivalence property between two systems involving two distinct identities that the attacker is unable to distinguish. As stated in previous sections, we analyze anonymity for both equivalences properties (i.e., trace and observational equivalence).

5.3.3 Results of the Analysis of PQ-WireGuard

The results of the analysis of PQ-WireGuard are depicted in Tables 5.12 on page 137 and 5.12 on page 137. For each analyzed security property, we compute its corresponding security formula. To simplify notation and enhance result readability, we denote the atomic capability for the reveal of the key using the key symbol itself, thus the set of the considered atomic capabilities is $\Gamma = \{\mathsf{psk}, \mathsf{s}_\mathsf{i}^\mathsf{pq}, \mathsf{s}_\mathsf{r}^\mathsf{pq}, \mathsf{r}_\mathsf{i}, \mathsf{r}_\mathsf{r}, \mathsf{r}_\mathsf{e}, \sigma_\mathsf{i}, \sigma_\mathsf{r}, \mathsf{e}_\mathsf{i}^\mathsf{pq}\}$ with respect to the notation given in Table 5.1. This analysis exclusively considers key-reveal atomic capabilities, excluding key modification capabilities. All our SAPIC$^+$ files can be accessed online [LMRT25].

We identify a bug in the PQ-WireGuard symbolic model from [HNS$^+$21] due to a discrepancy between the protocol description in the paper [HNS$^+$21] and its TAMARIN model [HNS$^+$20]. Specifically, in Algorithm 2. (ligne 12) of [HNS$^+$21], `ct2` is the ciphertext resulting from the encapsulation of the initiator's ephemeral key `epki`. This ciphertext is included in the key derivation chain in Table II in [HNS$^+$21] (which corresponds to step 6 of Table 5.14 on page 139). We verified the computational proof in the paper and confirm that it aligns with the protocol specifications, yet, the symbolic model does not. In line 221 of their TAMARIN code given in [HNS$^+$20], `sct2 = aenc{ka}pkI` represents the ciphertext resulting from the encapsulation of the initiator's static public key `pkI`, which is included in the derivation chain in subsequent lines. In contrast, `ect = aenc{k}pekI`, the ciphertext from encapsulating the initiator's ephemeral key `pekI` is never included in the key derivation chain. While this might appear to be a minor modeling error, its impact is significant: it transforms the lemma `UKS_on_responder_resistance` that was verified under the flawed model into a falsified lemma, as it enables an unknown key-share attack. This attack scenario, which we also identified in our model, will be explained in greater details later. In fact, the reason the attack disappears when the ciphertext (resulting from encapsulating the initiator static key) is included in the session key derivation chain is that the ciphertext (modeled as a standard public key encryption) remains bound to the static key. By incorporating it into the derivation chain, the session key itself also becomes bound to the initiator's static key. Thus, an honest responder cannot be tricked into establishing a session key with an initiator possessing a different key.

Unknown-Key-Share attacks. The security formula expressing minimal defensive models corresponding to UUKS resistance on the Initiator's side is $\mathsf{psk} \vee (\sigma_\mathsf{i} \wedge \mathsf{s}_\mathsf{i}^\mathsf{pq}) \vee (\mathsf{s}_\mathsf{i}^\mathsf{pq} \wedge \mathsf{r}_\mathsf{r}) \vee (\mathsf{e}_\mathsf{i}^\mathsf{pq} \wedge \mathsf{r}_\mathsf{e})$ which can be expressed as a disjunction of minimal offensive models as $(\mathsf{psk} \wedge \mathsf{s}_\mathsf{i}^\mathsf{pq} \wedge \mathsf{e}_\mathsf{i}^\mathsf{pq}) \vee (\mathsf{psk} \wedge \mathsf{s}_\mathsf{i}^\mathsf{pq} \wedge \mathsf{r}_\mathsf{e}) \vee (\mathsf{psk} \wedge \sigma_\mathsf{i} \wedge \mathsf{r}_\mathsf{r} \wedge \mathsf{e}_\mathsf{i}^\mathsf{pq}) \vee (\mathsf{psk} \wedge \sigma_\mathsf{i} \wedge \mathsf{r}_\mathsf{r} \wedge \mathsf{r}_\mathsf{e})$. Every offensive model incorporates both static keys and randomness generated during the session execution. Consequently, if either all static and pre-shared keys are compromised or if the randomness source is compromised, the protocol maintains UUKS resistance for the Initiator. This is interesting because the MEX scenario where all static keys and pre-shared keys are compromised, in our reading of [HNS+21], was excluded even for UKS attacks. An attacker possessing all static and pre-shared keys can impersonate peers, either by initiating or responding to sessions, but cannot force an honest Initiator to establish a session key with an honest Responder without their consent, even given full compromise of static and pre-shared keys. However, the protocol does not resist UKS attacks on the Initiator's side under any combination of compromised static keys and randomness.

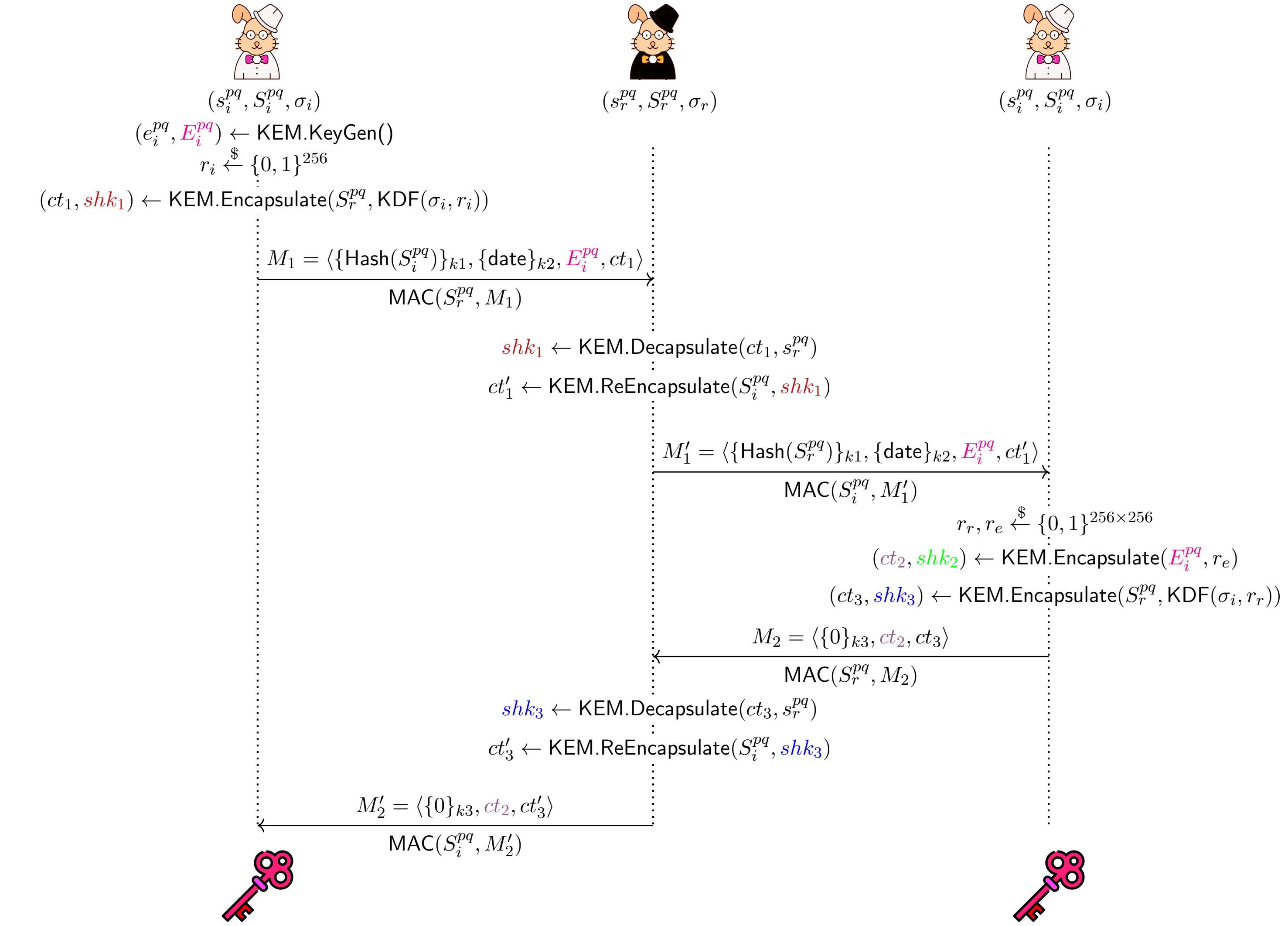
The same reasoning applies to the UUKS resistance on the Responder's side whose security formula is $\mathsf{psk} \vee (\sigma_\mathsf{i} \wedge \mathsf{s}_\mathsf{r}^\mathsf{pq}) \vee (\sigma_\mathsf{r} \wedge \mathsf{r}_\mathsf{i}) \vee (\mathsf{e}_\mathsf{i}^\mathsf{pq} \wedge \mathsf{r}_\mathsf{e})$. In [HNS+21], the authors claimed that the UUKS attack on the Responder is not possible: "*We prove that unilateral UKS on the responder is not possible*", and stated that "*Consequently, bilateral UKS is also not possible*". The first claim about the proof is not true because of the bug in their model we pointed out earlier in this section. For the second claim, even when UUKS attacks cannot be mounted against the Responder, this does not mean that BUKS attacks are impossible. We discovered a counterexample: Using the default pre-shared key $\mathsf{psk} = \mathsf{Hash}(\mathsf{S}_\mathsf{i}^\mathsf{pq} \oplus \mathsf{S}_\mathsf{r}^\mathsf{pq})$, we confirmed in Tamarin (using the `xor` builtin) that UUKS attacks are not possible against both the initiator or the responder. However, a BUKS attack does exist, as described below.

The security formula for the BUKS resistance property is $\mathsf{e}_\mathsf{i}^\mathsf{pq} \vee \mathsf{r}_\mathsf{e}$ as depicted in Table 5.12. BUKS resistance does not hold under the MEX scenario involving compromised randomness, contrary to the resistance requirement claimed by the protocol authors. An instantiation of an attack scenario is described in Figure 5.1 for the default pre-shared key configuration. The attack proceeds as follows: Alice, acting as an honest Initiator with a static public key $\mathsf{S}_\mathsf{i}^\mathsf{pq}$ and sharing a $\mathsf{psk} = \mathsf{Hash}(\mathsf{S}_\mathsf{i}^\mathsf{pq} \oplus \mathsf{S}_\mathsf{r}^\mathsf{pq})$ with the dishonest Eve with $\mathsf{S}_\mathsf{r}^\mathsf{pq}$ as public static key, generates an ehemeral key $\mathsf{e}_\mathsf{i}^\mathsf{pq}$, encapsulates the public key of Eve to abtain the shared secret shk_1, and sends an `InitHello` message. Eve receives Alice's message, decapsulates the received ciphertext with her secret key to obtain shk_3, re-encapsulates shk_1 using Alice's public key, and initiates a new session with Alice (consistent with WireGuard/PQ-WireGuard's dual-role design) by sending an `InitHello` message with the same psk, $\mathsf{e}_\mathsf{i}^\mathsf{pq}$, and the shared secret shk_1. When Alice receives Eve's new message, she processes it as a Responder by encapsulating Eve's public key to obtain the shared secret shk_1, and her own ephemeral key $\mathsf{e}_\mathsf{i}^\mathsf{pq}$ from the initial session to obain the shared secret shk_2, then transmits a `RespHello` response. Eve, possessing either the compromised randomness or the secret key corresponding the ephemeral key $\mathsf{e}_\mathsf{i}^\mathsf{pq}$ (per the security formula), completes the attack by decapsulating with her own key and re-encapsulating with Alice's public key. This results in Alice erroneously establishing a session key with herself while believing she has authenticated communications with Eve.

Attacks leveraging re-encapsulation in KEM-based constructions are termed *re-encapsulation attacks* [CDM24]. These attacks are not applicable across all KEMs, as not all KEM schemes permit re-encapsulation. For example, if the shared secrets (outputs of encapsulation) in PQ-WireGuard were cryptographically bound to their corresponding encapsulated public static keys, the aforementioned attack would be infeasible. The PQ-WireGuard authors implemented their protocol using Classic McEliece, which as mentioned in [CDM24] and based on the work of [GMP21], lacks this binding property, thereby enabling the described attack in practice. Since the shared secrets are not bound to static keys in this case, the derived session key only binds the protocol participants through the default pre-shared key. Crucially, the computed key material contains no distinguishing markers between the Initiator and Responder roles. This absence of role differentiation enables the BUKS attack we identified, which exploits two concurrent protocol sessions between the same participants with inverted roles. In WireGuard, these attacks are mitigated since the Diffie-Hellman products binds the shared secrets to the corresponding static keys.

To address BUKS vulnerabilities in PQ-WireGuard, we modify the key derivation process to incorporate the concatenation of the Initiator's and the Responder's public static keys respectively, i.e., $\mathsf{Hash}(\mathsf{S}_\mathsf{i}^\mathsf{pq} \mathbin{||} \mathsf{S}_\mathsf{r}^\mathsf{pq})$. This fix establishes an explicit cryptographic binding between the session keys and the protocol participants, independent of the underlying KEM's properties. It also explicitly encodes the initiator-responder distinction in the derived key material to prevent the role confusion exploited in BUKS attacks.

Session Uniqueness. The results of our analysis confirm the session uniqueness results of [HNS$^+$21] for both protocol participants. On the Initiator side, session key uniqueness stems from integrating the Initiator's ephemeral key together with the shared secret produced by encapsulating the Responder's public


$(s_i^{pq}, S_i^{pq}, \sigma_i)$
$(s_r^{pq}, S_r^{pq}, \sigma_r)$
$(s_i^{pq}, S_i^{pq}, \sigma_i)$
$(e_i^{pq}, E_i^{pq}) \leftarrow \mathsf{KEM.KeyGen}()$
$r_i \xleftarrow{\$} \{0,1\}^{256}$
$(ct_1, shk_1) \leftarrow \mathsf{KEM.Encapsulate}(S_r^{pq}, \mathsf{KDF}(\sigma_i, r_i))$
$M_1 = \langle \{\mathsf{Hash}(S_i^{pq})\}_{k1}, \{\mathsf{date}\}_{k2}, E_i^{pq}, ct_1 \rangle$
$\mathsf{MAC}(S_r^{pq}, M_1)$
$shk_1 \leftarrow \mathsf{KEM.Decapsulate}(ct_1, s_r^{pq})$
$ct_1' \leftarrow \mathsf{KEM.ReEncapsulate}(S_i^{pq}, shk_1)$
$M_1' = \langle \{\mathsf{Hash}(S_r^{pq})\}_{k1}, \{\mathsf{date}\}_{k2}, E_i^{pq}, ct_1' \rangle$
$\mathsf{MAC}(S_i^{pq}, M_1')$
$r_r, r_e \xleftarrow{\$} \{0,1\}^{256\times256}$
$(ct_2, shk_2) \leftarrow \mathsf{KEM.Encapsulate}(E_i^{pq}, r_e)$
$(ct_3, shk_3) \leftarrow \mathsf{KEM.Encapsulate}(S_r^{pq}, \mathsf{KDF}(\sigma_i, r_r))$
$M_2 = \langle \{0\}_{k3}, ct_2, ct_3 \rangle$
$\mathsf{MAC}(S_r^{pq}, M_2)$
$shk_3 \leftarrow \mathsf{KEM.Decapsulate}(ct_3, s_r^{pq})$
$ct_3' \leftarrow \mathsf{KEM.ReEncapsulate}(S_i^{pq}, shk_3)$
$M_2' = \langle \{0\}_{k3}, ct_2, ct_3' \rangle$
$\mathsf{MAC}(S_i^{pq}, M_2')$

- $k1 : KDF_2((KDF_1((ck, E_i^{pq})), shk_1))$
- $k2 : KDF_2((KDF_1((KDF_1((ck, E_i^{pq})), shk_1)), psk))$
- $k3 : KDF_3((KDF_1((KDF_1((KDF_1((KDF_1((KDF_1((KDF_1((ck, E_i^{pq})), shk_1)), psk)), ct_2)), shk_2)), shk_3)), psk))$
- 🔑 $: KDF_2(KDF_1((KDF_1((KDF_1((KDF_1((KDF_1((KDF_1((KDF_1((ck, E_i^{pq})), shk_1)), psk)), ct_2)), shk_2)), shk_3)), psk)))$

Figure 5.1: BUKS attack on PQ-Wireguard.

static key into the session key derivation process. This ensures that any honest Initiator properly generating their secrets can have confidence in session key uniqueness regardless of potential value reuse by the Responder. The Responder's session uniqueness guarantee similarly follows from incorporating multiple derivation inputs: the shared secret from encapsulating the Initiator's public static key, the share secret from encapsulating the Initiator's ephemeral key, and the associated ciphertext.

Anonymity. Likewise to WireGuard, our analysis show that neither the Initiator nor the Responder is anonymous when peer static keys are considered public knowledge. This vulnerability persists in the current context due to the MAC being computed using the peers' public keys. In Chapter 4, two fixes are proposed to ensure anonymity for WireGuard: one based on the pre-shared key psk, and another one based on an ECDH shared secret $\mathsf{dh}_{\mathsf{s_i s_r}}$. To reach anonymity in PQ-WireGuard, we consider the fix based on the pre-shared key, as the one based on $\mathsf{dh}_{\mathsf{s_i s_r}}$ is not applicable in the post-quantum case. The fix consists in computing the MAC value m1 in InitHello and RespHello, using the pre-shared key as the MAC key, instead of the static KEM key of the other party. This fix presents a computational challenge: when receiving an InitHello message, the Responder cannot immediately determine which pre-shared key from its database should be used to verify the incoming message. Consequently, the Responder must exhaustively test all available keys, resulting in significant computational overhead. To avoid this issue, we change the order of the Responder's operations upon the receipt of InitHello: the Responder has to perform KEM.Decaps and AEAD.Dec operations before m1 verification because the decryption of the static field allows them to identify the Initiator and retrieve the corresponding pre-shared key. Note that the original order is only inherited from the initial WireGuard's design. As explained in [Don17], the MAC check is meant protect the Responder against DoS attacks: if the verification fails, the Responder is prevented from computing costly ECDH, and AEAD.Dec computations. We note however that this protection is based on the strong assumption that the Initiator's and the Responder's static keys (used as MAC keys in WireGuard) are *secret*. Without this assumption, DoS attacks against the Responder are *realistic*: knowledge of the static public keys implies the ability to compute an arbitrary InitHello message with a correct MAC field, implying ECDH and AEAD.Dec computations.

Meanwhile, on the Initiator's side, no modification is required for RespHello message reception: the Initiator has the knowledge of which pre-shared key to use and checks the message authentication code m1 from the RespHello message, using the received sid_i field. For completeness, the InitHello message consumption is detailed in Table 5.11.

Message Agreement. The security formula for the agreement on the message `InitHello` is psk. When using the default pre-shared key $\mathsf{Hash}(\mathsf{S}_\mathsf{i}^{\mathsf{pq}} \oplus \mathsf{S}_\mathsf{r}^{\mathsf{pq}})$, and given that public static keys are publicly known, any entity can generate valid messages that will be accepted, processed, and responded to by the Responder. This creates a viable vector for denial-of-service attacks against the protocol. While PQ-WireGuard explicitly excludes this scenario through its assumption of secret public keys, this represents a potential design vulnerability. In practice, an attacker could be a protocol participant who has communicated with multiple peers, and frequent key rotation, WireGuard's mitigation for public key exposure, becomes challenging. WireGuard's 32-byte static keys facilitate such rotations, and according to WireGuard's whitepaper "*it is useful for transferring keys through a variety of different mediums*" [Don20]. In contrast PQ-WireGuard employs Classic McEliece public keys whose substantial size complicates this approach. Conversely, when a non-default psk is used, the Initiator achieves authentication of their first message. In our fix of the protocol, we used a non-default psk.

However, the KCI attack remains possible on the first message, which is the same conclusion as for WireGuard. Yet, a KCI attack cannot lead to a complete protocol execution. The security formula for the agreement on the `RespHello` message is $\mathsf{psk} \wedge (\mathsf{s}_\mathsf{r}^{\mathsf{pq}} \vee \mathsf{r}_\mathsf{i}) \wedge (\mathsf{s}_\mathsf{r}^{\mathsf{pq}} \vee \sigma_\mathsf{i})$, meaning an attacker with only the Initiator's static material cannot perform a KCI attack unless they have either compromised both the Initiator's randomness and long-term randomness, or compromised the Responder's static secret key. From these security formulas (expressed as conjunctions of minimal defensive models), we conclude that agreement on the second message holds for PQ-WireGuard under any MEX scenarios combining secrets and randomness, except those combinations appearing in defensive models, i.e., any combination that excludes psk or $\{\mathsf{s}_\mathsf{r}^{\mathsf{pq}}, \mathsf{r}_\mathsf{i}\}$ or $\{\mathsf{s}_\mathsf{r}^{\mathsf{pq}} \vee \sigma_\mathsf{i}\}$.

The security formula for the `Confirm` message is $\mathsf{psk} \wedge (\mathsf{s}_\mathsf{i}^{\mathsf{pq}} \vee \mathsf{r}_\mathsf{r}) \wedge (\mathsf{s}_\mathsf{i}^{\mathsf{pq}} \vee \sigma_\mathsf{r})$ showing a complete symmetry with the security formula on `RespHello`. Therefore, the same security arguments apply by simply inverting the roles. What is interesting to emphasize is that this symmetry is achieved in WireGuard's case which is based on the ECDH products, and it is also achieved when using KEMs. A direct comparison of the security formulas for WireGuard and PQ-WireGuard reveals their fundamental equivalence when excluding the term $\mathsf{dh}_{\mathsf{s_i s_r}}$ from WireGuard's security formula, as no analogous compromise case was considered for PQ-WireGuard. Both formulas provide identical security guarantees which precisely aligns with the security goals established by the PQ-WireGuard authors in their protocol design.

Key Secrecy and Forward Secrecy. The security formula for the session key secrecy from the Initiator's perspective is $\mathsf{psk} \wedge (\mathsf{s}_\mathsf{r}^\mathsf{pq} \vee \mathsf{r}_\mathsf{i}) \wedge (\mathsf{s}_\mathsf{r}^\mathsf{pq} \vee \sigma_\mathsf{i})$. To violate this property, an attacker must compromise either: (1) both the pre-shared key psk and the Responder's static secret key $\mathsf{s}_\mathsf{r}^\mathsf{pq}$, or (2) psk along with both the long-term randomness σ_i and the Initiator's session randomness r_i. This security formula matches WireGuard's formula when treating the long-term randomness as a static value. An equivalent formula exists from the Responder's perspective: $\mathsf{psk} \wedge (\mathsf{s}_\mathsf{i}^\mathsf{pq} \vee \mathsf{r}_\mathsf{r}) \wedge (\mathsf{s}_\mathsf{i}^\mathsf{pq} \vee \sigma_\mathsf{r})$. Both formulas provide resistance against MEX attack scenarios involving full randomness compromise, consistent with the analysis of [HNS$^+$21]. Notably, these security formulas for key secrecy coincide exactly with those for messages agreement. This means that session key secrecy from the Initiator's perspective is guaranteed if and only if the Responder authentication with reguard to the Initiator is satisfied, and peers have agreed on the session key. The same holds symmetrically for the Responder's perspective.

The mutual key secrecy formula is $\mathsf{psk} \wedge (\mathsf{s}_\mathsf{i}^\mathsf{pq} \vee \mathsf{r}_\mathsf{r}) \wedge (\mathsf{s}_\mathsf{i}^\mathsf{pq} \vee \sigma_\mathsf{r}) \wedge (\mathsf{s}_\mathsf{r}^\mathsf{pq} \vee \mathsf{r}_\mathsf{i}) \wedge (\mathsf{s}_\mathsf{r}^\mathsf{pq} \vee \sigma_\mathsf{i}) \wedge (\mathsf{e}_\mathsf{i}^\mathsf{pq} \vee \mathsf{r}_\mathsf{e})$. This formula shows that the minimal defensive models for mutual secrecy encompass those from both peers perspectives, that is, mutual secrecy holds if either the Initiator-side secrecy or the Responder-side secrecy holds. If the mutual key secrecy formula consisted solely of the conjunction of the peers secrecy formulas, i.e., $\mathsf{psk} \wedge (\mathsf{s}_\mathsf{i}^\mathsf{pq} \vee \mathsf{r}_\mathsf{r}) \wedge (\mathsf{s}_\mathsf{i}^\mathsf{pq} \vee \sigma_\mathsf{r}) \wedge (\mathsf{s}_\mathsf{r}^\mathsf{pq} \vee \mathsf{r}_\mathsf{i}) \wedge (\mathsf{s}_\mathsf{r}^\mathsf{pq} \vee \sigma_\mathsf{i})$, post-forward secrecy would not be achieved. Specifically, this would permit a minimal offensive model comprising only static and pre-shared keys $\mathsf{psk} \wedge \mathsf{s}_\mathsf{i}^\mathsf{pq} \wedge \mathsf{s}_\mathsf{r}^\mathsf{pq}$. However, the existence of the minimal defensive model $(\mathsf{e}_\mathsf{i}^\mathsf{pq} \vee \mathsf{r}_\mathsf{e})$ within the security formula that incorporates both the ephemeral key and session-generated randomness, introduces a protection against forward secrecy violations.

Strong Key Secrecy. The security formula for strong mutual key secrecy is $\mathsf{psk} \wedge (\mathsf{s}_\mathsf{r}^\mathsf{pq} \vee \mathsf{r}_\mathsf{i}) \wedge (\mathsf{s}_\mathsf{r}^\mathsf{pq} \vee \sigma_\mathsf{i})$. This formula provides stronger guarantees than the previously defined mutual secrecy. Notably, this security condition exactly matches the formula for the Initiator-side key secrecy, establishing that strong key secrecy is achieved precisely when the Initiator-side secrecy is satisfied.

Long-Term Randomness and Non-Standard KEMs. PQ-WireGuard incorporates long-term randomness alongside ephemeral randomness in the KEM to protect against MEX attacks involving full disclosure of ephemeral secrets and protocol randomness. Note that long-term randomness is treated equivalently to static keys. The security formulas in Tables 5.12 and 5.13, show that for every security property the pre-shared key is a minimal defensive model, meaning that no security property can be breached without its compromise. While a secret pre-shared key prevents MEX attacks under complete randomness compromise, the default pre-shared key's secrecy represents a strong assumption. Consequently, without the additional long-term randomness, the protocol's resistance to MEX attacks during full randomness compromise remains uncertain. In addition, the incorporation of long-term randomness creates practical implementation difficulties. This approach requires direct manipulation of the randomness input for the key encapsulation mechanism, contrary to safe standard KEM implementations [lib] where encapsulation under a public key is inherently probabilistic and does not permit user-controlled randomness.

In addition, the authors construct an IND-CPA KEM called Dagger based on SABER [BBD+20], a known KEM, to have messages small enough to avoid IP fragmentation. We stress that relying on such unconventionnal constructions can have security risks since it restricts the choices of libraries that can be used, forcing a developer to re-implement the KEM.

Based on our prior recommendation to eliminate default pre-shared keys and rely exclusively on secret pre-shared keys under the assumption of publicly known static keys, we now propose removing long-term randomness from the protocol specification. This modification enables a generic protocol design that operates independently of the non-standard KEM implementations originally proposed.

Summing Up: PQ-WireGuard*

We consider all previously suggested modifications into an updated variant of the protocol, which we refer to as PQ-WireGuard*, described in Tables 5 and 5.14. Specifically, our modifications are the following:

1. Inclusion of the hashed concatenation of the Initiator's public static key and the Responder's public static key $\mathsf{Hash}(\mathsf{S}_\mathsf{i}^\mathsf{pq} \mid\mid \mathsf{S}_\mathsf{r}^\mathsf{pq})$ into the session key derivation chain,
2. Replacement of the default pre-shared key with a secret pre-shared key,
3. Incorporation of the pre-shared key into the MAC computations, and
4. Elimination of long-term randomness.

The resulting security formulas of the analysis of PQ-WireGuard* are depicted in Tables 5.12 and 5.13 alongside prior analyses. The security formulas confirm that UKS attacks are now infeasible due to the explicit binding of static public keys in the key derivation process, and the introduction of a clear role differentiation between the Initiator and the Responder. Anonymity properties are successfully verified for both participants, though anonymity cannot be maintained in the MEX attack scenarios involving full compromise of ephemeral keys and all protocol-generated randomness. We observe that the minimal offensive models for the Responder's anonymity can also compromise the Initiator's anonymity. However, the Initiator's anonymity can be compromised without affecting the Responder's anonymity. All other security properties maintain their original formulations, with the sole modification being the removal of long-term randomness terms.

Algorithm 5 PQ-WireGuard⋆ handshake messages.

```
1. InitHello construction
    input: $s_i^{pq}, S_i^{pq}, S_r^{pq}$
    $\mathsf{sid}_i \xleftarrow{\$} \{0,1\}^{32}$
    $(\mathsf{e}_i^{pq}, \mathsf{E}_i^{pq}) \leftarrow \mathsf{KEM.gen}()$
    $(\mathsf{ct}_1, \mathsf{shk}_1) \leftarrow \mathsf{KEM.Encaps}(\mathsf{S}_r^{pq})$
    $\mathsf{static} \leftarrow \mathsf{AEAD.Enc}(\kappa_3,\ 0,\ H_3,\ \mathsf{Hash}(\mathsf{S}_i^{pq}))$
    $\mathsf{time} \leftarrow \mathsf{AEAD.Enc}(\kappa_4,\ 0,\ H_4,\ \mathsf{now}())$
    $\mathsf{inner} \leftarrow \mathsf{type} \mid\mid \mathsf{sid}_i \mid\mid \mathsf{E}_i^{pq} \mid\mid \mathsf{ct}_1 \mid\mid \mathsf{static} \mid\mid \mathsf{time}$
    $\mathsf{m1} \leftarrow \mathsf{MAC}(\mathsf{Hash}(\mathsf{lbl3} \mid\mid \mathsf{psk}),\ \mathsf{inner})$
    $\mathsf{m2} \leftarrow \mathsf{MAC}(\mathsf{cookie},\ \mathsf{inner} \mid\mid \mathsf{m1})$
    $\mathsf{InitHello} \leftarrow \mathsf{inner} \mid\mid \mathsf{m1} \mid\mid \mathsf{m2}$
2. RespHello construction
    input: $s_r^{pq}, S_r^{pq}, S_i^{pq}$
    $\mathsf{sid}_r \xleftarrow{\$} \{0,1\}^{32}$
    $(\mathsf{ct}_2, \mathsf{shk}_2) \leftarrow \mathsf{KEM.Encaps}(\mathsf{E}_i^{pq})$
    $(\mathsf{ct}_3, \mathsf{shk}_3) \leftarrow \mathsf{KEM.Encaps}(\mathsf{S}_i^{pq})$
    $\mathsf{empty} \leftarrow \mathsf{AEAD.Enc}(\kappa_9,\ 0,\ H_9,\ \varnothing)$

    $\mathsf{inner} \leftarrow \mathsf{type} \mid\mid \mathsf{sid}_r \mid\mid \mathsf{sid}_i \mid\mid \mathsf{ct}_2 \mid\mid \mathsf{ct}_3 \mid\mid \mathsf{empty}$
    $\mathsf{m1} \leftarrow \mathsf{MAC}(\mathsf{Hash}(\mathsf{lbl3} \mid\mid \mathsf{psk}),\ \mathsf{inner})$
    $\mathsf{m2} \leftarrow \mathsf{MAC}(\mathsf{cookie},\ \mathsf{inner} \mid\mid \mathsf{m1})$
    $\mathsf{RespHello} \leftarrow \mathsf{inner} \mid\mid \mathsf{m1} \mid\mid \mathsf{m2}$
```

5.4 Hybrid-WireGuard: Protocol and Analysis

This section presents the design of a new protocol called Hybrid-WireGuard. The protocol design builds upon WireGuard and PQ-WireGuard, with the objectives of achieving hybrid security and complying with post-quantum transition recommendations. We establish a formal security goal: the security formulas for all the defined security properties in the hybrid protocol must combine the security formulas of WireGuard and PQ-WireGuard⋆. Specifically, any minimal defensive model present in either constituent protocol must also be a minimal defensive model in the hybrid protocol. This requirement ensures: (1) preservation of all defensive models from both base protocols, and (2) the necessity for an attacker to simultaneously compromise both constituent protocols to attack the hybrid version. The section first details the protocol design, then we present the analysis results.

Protocol definition. We introduce the handshake, described in Tables 5.10 and 5.14, that hybridizes WireGuard: Hybrid-WireGuard combines ECDH secrets from the WireGuard handshake, with post-quantum KEM secrets from PQ-WireGuard*. To achieve our stated security goals, we propose a construction inspired from existing works (e.g., [FMJ24, KSH24, TTB+23]). In this case, the Initiator and the Responder hold static long-term ECDH and KEM key pairs $\big((\mathsf{s}_\mathsf{i}^\mathsf{c}, \mathsf{S}_\mathsf{i}^\mathsf{c}), (\mathsf{s}_\mathsf{i}^\mathsf{pq}, \mathsf{S}_\mathsf{i}^\mathsf{pq})\big)$ and $\big((\mathsf{s}_\mathsf{r}^\mathsf{c}, \mathsf{S}_\mathsf{r}^\mathsf{c}), (\mathsf{s}_\mathsf{r}^\mathsf{pq}, \mathsf{S}_\mathsf{r}^\mathsf{pq})\big)$ respectively. We also derive four ECDH secrets as in WireGuard, and three KEM shared secrets as in PQ-WireGuard*.

During key derivation, both the ECDH secrets and the KEM secrets from WireGuard and PQ-WireGuard* respectively, are concatenated and used in Hybrid-WireGuard (terms C_3, C_4, C_7, C_8 and term κ_3 in Table 5.14). In addition, we include the necessary modifications to ensure resistance against UKS attacks: we use the concatenation of the ECDH product from WireGuard, and the hash of the KEM public keys from PQ-WireGuard*, both used in the corresponding protocols to reach this resistance (terms C_4 and κ_4 in Table 5.14).

During the handshake, the static field results from an AEAD encryption of the Initiator's ECDH static public key in WireGuard, and the hash of the Initiator's KEM static public key in PQ-WireGuard* (considering that the public key would increase `InitHello` message size above the MTU limit [HNS+21]). In Hybrid-WireGuard, the static field is an encryption of the hash of the Initiator's concatenated ECDH and KEM public keys.

For anonymity, another tweak is needed in the case of Hybrid-WireGuard to ensure that breaking the property requires the compromise of both ECDH and KEM static keys. During the PQ-WireGuard* handshake, if an attacker compromises the ephemeral randomness used during the encapsulation against $\mathsf{S}_\mathsf{r}^\mathsf{pq}$ in `InitHello`, then they can intercept ct_1 to lookup the correct static KEM key and reveal the identity of the Responder. Similarly, if an attacker compromises the ephemeral randomness used during the encapsulation against $\mathsf{S}_\mathsf{i}^\mathsf{pq}$ in `RespHello`, then they can intercept ct_3 to lookup the correct static KEM key and reveal the identity of the Initiator. This vulnerability is inherited by Hybrid-WireGuard when sending the same ciphertexts ct_1 and ct_3 respectively in `InitHello` and `RespHello`, making the identity compromise only requires the lookup of the KEM static keys. To make up for this issue, we encrypt ct_1 and ct_3 with a symmetric encryption scheme, using as key the output of KDF_1 applied to the ECDH secret $\mathsf{dh}_{\mathsf{e_i s_r}}$ in the case of `InitHello`, and $\mathsf{dh}_{\mathsf{s_i e_r}}$ in the case of `RespHello`. Hence, instead of sending ct_1 and ct_3 in `InitHello` and `RespHello` respectively, the Initiator sends $\mathsf{ct}_1{}^{\mathrm{enc}}$ created on line 7 for `InitHello` in Table 5.10 and $\mathsf{ct}_3{}^{\mathrm{enc}}$ created on line 6 for `RespHello`. The goal of this tweak is that even in the case of ephemeral randomness compromise, the attacker would still need to lookup all possible ECDH and KEM keys at the same time. Consequently, compromising only the ECDH static key or only the KEM key, along with ephemeral randomness compromise, is not enough to break anonymity. Note that ephemeral randomness compromise does not have the same effect in the case of PQ-WireGuard, since the secret is a result of the deterministic encapsulation, using random coins generated from the combination of ephemeral and long-term secret randomness.

1. `InitHello` construction

```
input: $\mathsf{s_i^c}, \mathsf{S_i^c}, \mathsf{S_r^c}, \mathsf{s_i^{pq}}, \mathsf{S_i^{pq}}, \mathsf{S_r^{pq}}$
$\mathsf{sid}_i \xleftarrow{\$} \{0,1\}^{32}$
$(\mathsf{e_i^c}, \mathsf{E_i^c}) \leftarrow \mathsf{DH.gen}()$
$(\mathsf{e_i^{pq}}, \mathsf{E_i^{pq}}) \leftarrow \mathsf{KEM.gen}()$
$(\mathsf{ct_1}, \mathsf{shk_1}) \leftarrow \mathsf{KEM.Encaps}(\mathsf{S_r^{pq}})$
$\mathsf{static} \leftarrow \mathsf{AEAD.Enc}(\kappa_3,\ 0,\ H_3,\ \mathsf{Hash}(\mathsf{S_i^c} \mid\mid \mathsf{S_i^{pq}}))$
$\mathsf{time} \leftarrow \mathsf{AEAD.Enc}(\kappa_4,\ 0,\ H_4,\ \mathsf{now}())$
$\mathsf{ct_1}^{\mathsf{enc}} \leftarrow \mathsf{SE.Enc}(\mathsf{KDF}_1(\varnothing,\ \mathsf{dh}_{\mathsf{e_i s_r}}),\ 0,\ \mathsf{ct_1})$
$\mathsf{inner} \leftarrow \mathsf{type} \mid\mid \mathsf{sid}_i \mid\mid \mathsf{E_i^c} \mid\mid \mathsf{E_i^{pq}} \mid\mid \mathsf{ct_1}^{\mathsf{enc}} \mid\mid \mathsf{static} \mid\mid \mathsf{time}$
$\mathsf{m1} \leftarrow \mathsf{MAC}(\mathsf{Hash}(\mathsf{lbl3} \mid\mid \mathsf{psk}),\ \mathsf{inner})$
$\mathsf{m2} \leftarrow \mathsf{MAC}(\mathsf{cookie},\ \mathsf{inner} \mid\mid \mathsf{m1})$
$\mathsf{InitHello} \leftarrow \mathsf{inner} \mid\mid \mathsf{m1} \mid\mid \mathsf{m2}$
```

2. `RespHello` construction

```
input: $\mathsf{s_r^c}, \mathsf{S_r^c}, \mathsf{S_i^c}, \mathsf{s_r^{pq}}, \mathsf{S_r^{pq}}, \mathsf{S_i^{pq}}$
$\mathsf{sid}_r \xleftarrow{\$} \{0,1\}^{32}$
$(\mathsf{e_r^c}, \mathsf{E_r^c}) \leftarrow \mathsf{DH.gen}()$
$(\mathsf{ct_2}, \mathsf{shk_2}) \leftarrow \mathsf{KEM.Encaps}(\mathsf{E_i^{pq}})$
$(\mathsf{ct_3}, \mathsf{shk_3}) \leftarrow \mathsf{KEM.Encaps}(\mathsf{S_i^{pq}})$
$\mathsf{empty} \leftarrow \mathsf{AEAD.Enc}(\kappa_9,\ 0,\ H_9,\ \varnothing)$

$\mathsf{ct_3}^{\mathsf{enc}} \leftarrow \mathsf{SE.Enc}(\mathsf{KDF}_1(\varnothing,\ \mathsf{dh}_{\mathsf{s_i e_r}}),\ 0,\ \mathsf{ct_3})$
$\mathsf{inner} \leftarrow \mathsf{type} \mid\mid \mathsf{sid}_r \mid\mid \mathsf{sid}_i \mid\mid \mathsf{E_r^c} \mid\mid \mathsf{ct_2} \mid\mid \mathsf{ct_3}^{\mathsf{enc}} \mid\mid \mathsf{empty}$
$\mathsf{m1} \leftarrow \mathsf{MAC}(\mathsf{Hash}(\mathsf{lbl3} \mid\mid \mathsf{psk}),\ \mathsf{inner})$
$\mathsf{m2} \leftarrow \mathsf{MAC}(\mathsf{cookie},\ \mathsf{inner} \mid\mid \mathsf{m1})$
$\mathsf{RespHello} \leftarrow \mathsf{inner} \mid\mid \mathsf{m1} \mid\mid \mathsf{m2}$
```

Table 5.10: Hybrid-WireGuard handshake messages.

Symbolic Analysis Results. For our verification of Hybrid-WireGuard's security properties, we directly used the obtained security formulas derived from prior analyses of both WireGuard and PQ-WireGuard*. Consistent with the design requirement that the hybrid protocol's security formulas must incorporate all minimal defensive models from both constituent protocols, we verified a single Tamarin lemma (for trace-based properties) representing the logical conjunction of these two security formulas. For the verification of privacy properties (strong secrecy and anonymity), we used only ProVerif, as DeepSec lacks support for the exponentiation equations required to model the Diffie-Hellman keys. The results of our symbolic analysis for the Hybrid-WireGuard are presented side-by-side with other protocols in Table 5.12 for Unknown-Key-Share attacks resistance, session uniqueness, anonymity and message agreement, and in Table 5.13 for all secrecy properties. We put forward that a necessary and sufficient condition to break a given property in our proposed Hybrid-WireGuard construction is to break the same property for WireGuard and for PQ-WireGuard*, with exactly the same set of keys involved. The results depicted in Tables 5.12 and 5.13 reveal that all security formulas (the Initiator's anonymity excluded) are formed by the logical conjunction of WireGuard's security formula and PQ-WireGuard* 's security formula. For example, the security formula for the agreement on `RecHello` message is $\mathsf{psk} \wedge (\mathsf{s}_\mathsf{r}^\mathsf{c} \vee \mathsf{e}_\mathsf{i}^\mathsf{c}) \wedge (\mathsf{dh}_{\mathsf{s_i s_r}} \vee \mathsf{s}_\mathsf{i}^\mathsf{c} \vee \mathsf{s}_\mathsf{r}^\mathsf{c}) \wedge (\mathsf{s}_\mathsf{r}^\mathsf{pq} \vee \mathsf{r_i})$ which is the conjunction of $\mathsf{psk} \wedge (\mathsf{s}_\mathsf{r}^\mathsf{c} \vee \mathsf{e}_\mathsf{i}^\mathsf{c}) \wedge (\mathsf{dh}_{\mathsf{s_i s_r}} \vee \mathsf{s}_\mathsf{i}^\mathsf{c} \vee \mathsf{s}_\mathsf{r}^\mathsf{c})$ (WireGuard's security formula for agreement on `RecHello` message), and $\mathsf{psk} \wedge (\mathsf{s}_\mathsf{r}^\mathsf{pq} \vee \mathsf{r_i})$ (PQ-WireGuard*'s security formula for the same security property). Regarding the Initiator's anonymity, the obtained result is "better" than the conjunction of both defensive models. Specifically, we eliminated attack vectors existing in PQ-WireGuard* to achieve the Initiator-Responder symmetry. While the removal of the minimal defensive models $\mathsf{r_i}$ and $\mathsf{s}_\mathsf{i}^\mathsf{pq}$ from the security formulas may appear unjustified, it becomes justified when the minimal defensive model is itself a minimal offensive model - representing an inherent attack vector.

We note that the Hybrid protocol's design involved continuous iteration between the protocol design and formal analysis. These results were not obtained from the first proposed version, confirming the critical role of formal analysis during protocol design.

5.5 Conclusion and Future Work

Using the automatic verification tools ProVerif, DeepSec, and Tamarin, we analyzed the security of PQ-WireGuard. The analysis showed that the protocol can be fixed to reach anonymity and resilience against Unknown Key-Share attacks. We therefore proposed an improved version, PQ-WireGuard*, for which these properties are ensured, without degrading other properties already reached by PQ-WireGuard (agreement, key secrecy, mutual secrecy, forward secrecy, session uniqueness). From WireGuard and PQ-WireGuard*, we construct Hybrid-WireGuard, that offers the best of the two worlds (classic and post-quantum), and we formally proved the security of our construction. We reach a hybridization target, because we show that an attack scenario against a security property of Hybrid-WireGuard is indeed a combination of an attack on PQ-WireGuard* and an attack on the original WireGuard.

Another direction for the symbolic analysis of PQ-WireGuard and Hybrid-WireGuard involves identifying the minimal binding properties required for the

KEMs used in our protocol in orded to avoid other re-encapsulation attacks. The Tamarin library proposed by [CDM24] can be used to automatically derive the minimal binding properties required from the KEM in order for the protocol to meet its security goals.

Rosenpass [VLZ+24] is an implementation of PQ-WireGuard [HNS+21], which aims to provide peers with a post-quantum shared secret. This post-quantum shared secret can be used as a pre-shared key when using the WireGuard protocol, that is, providing hybridization. While symbolic analyses with ProVerif exist for Rosenpass, their results (which simply indicate that security properties are verified without examining compromise scenarios) preclude detailed comparison with our protocol. Specifically, the available analysis does not enable direct comparison between our hybridization results and theirs. Analyzing Rosenpass within our framework would facilitate such protocol comparisons.

Our work also shows the limitations of using post-quantum primitives in WireGuard, specifically related to the sizes of the exchanged messages. As WireGuard does not handle fragmentation, the packets' sizes must not exceed the IPv6 MTU limit (1280 bytes). Note that this problem is inherent to the post-quantum transition of any protocol that does not handle fragmentation. Hence we have two choices. First is to allow additional messages in the handshake. Clearly, this would render WireGuard non-competitive compared to other VPNs: one main advantage of WireGuard is precisely this two-message handshake. Second, as in [HNS+21], is to keep this two-message handshake design. In this case, we become limited for KEM algorithms, which is why Classic McEliece is the only KEM suitable for the static keys (even if we only aim L1 security). For ephemeral keys, [HNS+21] tweaks an existing KEM construction for L3 security, while we argue that this is not a desired practice, so we restrict our design to trusted constructions and implementations. This limits us to use Kyber, with L1 security. We think that future research should analyze if L3 or L5 levels are reachable for two-messages handshakes.

WireGuard [Don17], anonymity not ensured

```
input: InitHello, $s_r^c$, $S_r^c$
parse InitHello as inner || m1 || m2
check m1 == MAC(Hash(lbl3 || $S_r^c$), inner)
parse inner as type || $sid_i$ || $E_i^c$ || static || time
compute $dh_{e_i s_r}$   // used to compute $\kappa_3$
$S_i^c \leftarrow$ AEAD.Dec($\kappa_3$, 0, static)
lookup Initiator public keys and psk using $S_i^c$
AEAD.Dec($\kappa_4$, 0, time)
```

PQ-WireGuard [HNS$^+$21], anonymity not ensured

```
input: InitHello, $s_r^{pq}$, $S_r^{pq}$
parse InitHello as inner || m1 || m2
check m1 == MAC(Hash(lbl3 || $S_r^{pq}$), inner)
parse inner as type || $sid_i$ || $E_i^{pq}$ || $ct_1$ || static || time
$shk_1 \leftarrow$ KEM.Decaps($s_r^{pq}$, $ct_1$)
h $\leftarrow$ AEAD.Dec($\kappa_3$, 0, static)   // h = Hash($S_i^{pq}$)
lookup Initiator public keys and psk using h
AEAD.Dec($\kappa_4$, 0, time)
```

PQ-WireGuard*, anonymity ensured

```
input: InitHello, $s_r^{pq}$, $S_r^{pq}$
parse InitHello as inner || m1 || m2
parse inner as type || $sid_i$ || $E_i^{pq}$ || $ct_1$ || static || time
$shk_1 \leftarrow$ KEM.Decaps($s_r^{pq}$, $ct_1$)
h $\leftarrow$ AEAD.Dec($\kappa_3$, 0, static)   // h = Hash($S_i^{pq}$)
lookup Initiator public keys and psk using h
check m1 == MAC(Hash(lbl3 || psk), inner)
AEAD.Dec($\kappa_4$, 0, time)
```

Hybrid-WireGuard, anonymity ensured

```
input: InitHello, $s_r^c$, $S_r^c$, $s_r^{pq}$, $S_r^{pq}$
parse InitHello as inner || m1 || m2
parse inner as type || $sid_i$ || $E_i^c$ || $E_i^{pq}$ || ${ct_1}^{enc}$ || static
|| time
compute $dh_{e_i s_r}$   // used to compute $\kappa_3$
$ct_1 \leftarrow$ SE.Dec($KDF_1(\varnothing, dh_{e_i s_r}), 0, {ct_1}^{enc}$)
$shk_1 \leftarrow$ KEM.Decaps($s_r^{pq}$, $ct_1$)
h $\leftarrow$ AEAD.Dec($\kappa_3$, 0, static)   // h = Hash($S_i^c$ || $S_i^{pq}$)
lookup Initiator public keys and psk using h
check m1 == MAC(Hash(lbl3 || psk), inner)
AEAD.Dec($\kappa_4$, 0, time)
```

Table 5.11: Responder's InitHello consumption by Responder.

Property	**WireGuard**	**PQ-WireGuard**	**PQ-WireGuard***	**Hybrid-WireGuard**
UKS attacks resistance (ProVerif, Tamarin)				
UUKS Initiator	✓	$psk \lor (\sigma_i \land s_i^{pq}) \lor (s_i^{pq} \land r_r) \lor (e_i^{pq} \land r_e)$	✓	✓
UUKS Responder	✓	$psk \lor (\sigma_i \land s_r^{pq}) \lor (\sigma_r \land r_i) \lor (e_i^{pq} \land r_e)$	✓	✓
Bilateral	✓	$e_i^{pq} \lor r_e$	✓	✓
Session uniqueness (ProVerif, Tamarin)				
Init./Resp.	✓	✓	✓	✓
Anonymity (DeepSec, ProVerif)				
Initiator	$psk \lor s_r^c \lor e_i^c$	✗	$psk \lor s_i^{pq} \lor s_r^{pq} \lor r_r \lor r_i$	$(psk \lor s_r^{pq} \lor r_r) \bigwedge (psk \lor s_r^c \lor e_i^c)$
Responder	$psk \lor s_i^c \lor e_r^c$	✗	$psk \lor s_r^{pq} \lor r_i$	$(psk \lor s_r^{pq} \lor r_i) \bigwedge (psk \lor s_i^c \lor e_r^c)$
Message agreement (ProVerif, Tamarin)				
`InitHello`	$dh_{s_i s_r} \lor s_i^c \lor s_r^c$	psk	psk	$(dh_{s_i s_r} \lor s_i^c \lor s_r^c) \bigwedge psk$
`RespHello`	$psk \land (s_r^c \lor e_i^c) \land (dh_{s_i s_r} \lor s_i^c \lor s_r^c)$	$psk \land (s_r^{pq} \lor r_i) \land (s_r^{pq} \lor \sigma_i)$	$psk \land (s_r^{pq} \lor r_i)$	$psk \land (s_r^c \lor e_i^c) \land (dh_{s_i s_r} \lor s_i^c \lor s_r^c) \bigwedge (s_r^{pq} \lor r_i)$
Confirm	$psk \land (s_i^c \lor e_r^c) \land (dh_{s_i s_r} \lor s_i^c \lor s_r^c)$	$psk \land (s_i^{pq} \lor r_r) \land (s_i^{pq} \lor \sigma_r)$	$psk \land (s_i^{pq} \lor r_r)$	$psk \land (s_i^c \lor e_r^c) \land (dh_{s_i s_r} \lor s_i^c \lor s_r^c) \bigwedge (s_i^{pq} \lor r_r)$

Table 5.12: Security formulas for UKS attack resistance, session uniqueness, anonymity and message agreement. ✓denotes a property is ensured even when the attacker has all the considered atomic capabilities, ✗denotes a property not ensured in the presence of a Dolev-Yao attacker. Key notations are from Table 5.1, with dedicated colors: blue for WireGuard, orange for PQ-WireGuard* and both colors for Hybrid-WireGuard. Note that for WireGuard, we use and complete the results from Chapter 4 as it allows to consider a fix we proposed for anonymity.

Property	**WireGuard**	**PQ-WireGuard**	**PQ-WireGuard***	**Hybrid-WireGuard**
Key secrecy (ProVerif, Tamarin)				
Initiator's point of view	$\mathsf{psk} \wedge (s_r^c \vee e_i^c) \wedge (dh_{s_i s_r} \vee s_i^c \vee s_r^c)$	$\mathsf{psk} \wedge (s_r^{pq} \vee r_i) \wedge (s_r^{pq} \vee \sigma_i)$	$\mathsf{psk} \wedge (s_r^{pq} \vee r_i)$	$\mathsf{psk} \wedge (s_r^c \vee e_i^c) \wedge (dh_{s_i s_r} \vee s_i^c \vee s_r^c) \bigwedge (s_r^{pq} \vee r_i)$
Responder's point of view	$\mathsf{psk} \wedge (s_i^c \vee e_r^c) \wedge (dh_{s_i s_r} \vee s_i^c \vee s_r^c)$	$\mathsf{psk} \wedge (s_i^{pq} \vee r_r) \wedge (s_i^{pq} \vee \sigma_r)$	$\mathsf{psk} \wedge (s_i^{pq} \vee r_r)$	$\mathsf{psk} \wedge (s_i^c \vee e_r^c) \wedge (dh_{s_i s_r} \vee s_i^c \vee s_r^c) \bigwedge (s_i^{pq} \vee r_r)$
Mutual key secrecy	$\mathsf{psk} \wedge (s_i^c \vee e_r^c) \wedge (s_r^c \vee e_i^c) \wedge (e_i^c \vee e_r^c) \wedge (dh_{s_i s_r} \vee s_i^c \vee s_r^c)$	$\mathsf{psk} \wedge (s_i^{pq} \vee r_r) \wedge (s_i^{pq} \vee \sigma_r) \wedge (s_r^{pq} \vee r_i) \wedge (s_r^{pq} \vee \sigma_i) \wedge (e_i^{pq} \vee r_e)$	$\mathsf{psk} \wedge (s_i^{pq} \vee r_r) \wedge (s_r^{pq} \vee r_i) \wedge (e_i^{pq} \vee r_e)$	$\mathsf{psk} \wedge (s_i^c \vee e_r^c) \wedge (s_r^c \vee e_i^c) \wedge (e_i^c \vee e_r^c) \wedge (dh_{s_i s_r} \vee s_i^c \vee s_r^c) \bigwedge (s_i^{pq} \vee r_r) \wedge (s_r^{pq} \vee r_i) \wedge (e_i^{pq} \vee r_e)$
Key strong secrecy (ProVerif, DeepSec)				
Strong key secrecy	$\mathsf{psk} \wedge (s_r^c \vee e_i^c) \wedge (dh_{s_i s_r} \vee s_i^c \vee s_r^c)$	$\mathsf{psk} \wedge (s_r^{pq} \vee r_i) \wedge (s_r^{pq} \vee \sigma_i)$	$\mathsf{psk} \wedge (s_r^{pq} \vee r_i)$	$\mathsf{psk} \wedge (s_r^c \vee e_i^c) \wedge (dh_{s_i s_r} \vee s_i^c \vee s_r^c) \bigwedge (s_r^{pq} \vee r_i)$
Key forward secrecy (ProVerif, Tamarin)				
Key forward secrecy	$\mathsf{psk} \wedge (s_i^c \vee e_r^c) \wedge (s_r^c \vee e_i^c) \wedge (e_i^c \vee e_r^c) \wedge (dh_{s_i s_r} \vee s_i^c \vee s_r^c)$	$\mathsf{psk} \wedge (s_i^{pq} \vee r_r) \wedge (s_i^{pq} \vee \sigma_r) \wedge (s_r^{pq} \vee r_i) \wedge (s_r^{pq} \vee \sigma_i) \wedge (e_i^{pq} \vee r_e)$	$\mathsf{psk} \wedge (s_i^{pq} \vee r_r) \wedge (s_r^{pq} \vee r_i) \wedge (e_i^{pq} \vee r_e)$	$\mathsf{psk} \wedge (s_i^c \vee e_r^c) \wedge (s_r^c \vee e_i^c) \wedge (e_i^c \vee e_r^c) \wedge (dh_{s_i s_r} \vee s_i^c \vee s_r^c) \bigwedge (s_i^{pq} \vee r_r) \wedge (s_r^{pq} \vee r_i) \wedge (e_i^{pq} \vee r_e)$

Table 5.13: Security formulas for key secrecy, strong key secrecy, mutual key secrecy, and key forward secrecy. Key notations are from Table 5.1, with dedicated colors: blue for WireGuard [Don17], orange for PQ-WireGuard* and both colors for Hybrid-WireGuard.

	k	C_k	κ_k	H_k
	1	$\mathsf{Hash}(\mathsf{lbl1})$	-	$\mathsf{Hash}(C_1 \Vert \mathsf{lbl2})$
WG [Don17]	2	$KDF_1(C_1,\ \mathsf{E_i^c})$	-	$\mathsf{Hash}(H_1 \Vert \mathsf{S_r^c})$
	3	$KDF_2(C_2,\ \mathsf{dh_{e_is_r}})[1]$	$KDF_2(C_2,\ \mathsf{dh_{e_is_r}})[2]$	$\mathsf{Hash}(H_2 \Vert \mathsf{E_i^c})$
	4	$KDF_2(C_3,\ \mathsf{dh_{s_is_r}})[1]$	$KDF_2(C_3,\ \mathsf{dh_{s_is_r}})[2]$	$\mathsf{Hash}(H_3 \Vert \mathsf{InitHello.static})$
	5	-	-	$\mathsf{Hash}(H_4 \Vert \mathsf{InitHello.time})$
	6	$KDF_1(C_4,\ \mathsf{E_r^c})$	-	$\mathsf{Hash}(H_5 \Vert \mathsf{E_r^c})$
	7	$KDF_1(C_6,\ \mathsf{dh_{e_ie_r}})$	-	-
	8	$KDF_1(C_7,\ \mathsf{dh_{s_ie_r}})$	-	-
PQ-WG [HNS+21]	2	$KDF_1(C_1,\ \mathsf{E_i^{pq}})$	-	$\mathsf{Hash}(H_1 \Vert \mathsf{S_r^{pq}})$
	3	$KDF_2(C_2,\ \mathsf{shk_1})[1]$	$KDF_2(C_2,\ \mathsf{shk_1})[2]$	$\mathsf{Hash}(H_2 \Vert \mathsf{E_i^{pq}})$
	4	$KDF_2(C_3,\ \mathsf{psk})[1]$	$KDF_2(C_3,\ \mathsf{psk})[2]$	$\mathsf{Hash}(H_3 \Vert \mathsf{InitHello.static})$
	5	-	-	$\mathsf{Hash}(H_4 \Vert \mathsf{InitHello.time})$
	6	$KDF_1(C_4,\ \mathsf{ct_2})$	-	$\mathsf{Hash}(H_5 \Vert \mathsf{ct_2})$
	7	$KDF_1(C_6,\ \mathsf{shk_2})$	-	-
	8	$KDF_1(C_7,\ \mathsf{shk_3})$	-	-
PQ-WG⋆ (Section 5.3.3)	2	$KDF_1(C_1,\ \mathsf{E_i^{pq}})$	-	$\mathsf{Hash}(H_1 \Vert \mathsf{Hash}(\mathsf{S_r^{pq}}))$
	3	$KDF_2(C_2,\ \mathsf{shk_1})[1]$	$KDF_2(C_2,\ \mathsf{shk_1})[2]$	$\mathsf{Hash}(H_2 \Vert \mathsf{E_i^{pq}})$
	4	$KDF_2(C_3,\ \mathsf{Hash}(\mathsf{S_i^{pq}} \Vert \mathsf{S_r^{pq}}) \Vert \mathsf{psk})[1]$	$KDF_2(C_3,\ \mathsf{Hash}(\mathsf{S_i^{pq}} \Vert \mathsf{S_r^{pq}}) \Vert \mathsf{psk})[2]$	$\mathsf{Hash}(H_3 \Vert \mathsf{InitHello.static})$
	5	-	-	$\mathsf{Hash}(H_4 \Vert \mathsf{InitHello.time})$
	6	$KDF_1(C_4,\ \mathsf{ct_2})$	-	$\mathsf{Hash}(H_5 \Vert \mathsf{ct_2})$
	7	$KDF_1(C_6,\ \mathsf{shk_2})$	-	-
	8	$KDF_1(C_7,\ \mathsf{shk_3})$	-	-
Hybrid-WG (Section 5.4)	2	$KDF_1(C_1,\ \mathsf{E_i^c} \Vert \mathsf{E_i^{pq}})$	-	$\mathsf{Hash}(H_1 \Vert \mathsf{Hash}(\mathsf{S_r^c} \Vert \mathsf{S_r^{pq}}))$
	3	$KDF_2(C_2,\ \mathsf{dh_{e_is_r}} \Vert \mathsf{shk_1})[1]$	$KDF_2(C_2,\ \mathsf{dh_{e_is_r}} \Vert \mathsf{shk_1})[2]$	$\mathsf{Hash}(H_2 \Vert \mathsf{E_i^c} \Vert \mathsf{E_i^{pq}})$
	4	$KDF_2(C_3,\ \mathsf{dh_{s_is_r}} \Vert \mathsf{Hash}(\mathsf{S_i^{pq}} \Vert \mathsf{S_r^{pq}}) \Vert \mathsf{psk})[1]$	$KDF_2(C_3,\ \mathsf{dh_{s_is_r}} \Vert \mathsf{Hash}(\mathsf{S_i^{pq}} \Vert \mathsf{S_r^{pq}}) \Vert \mathsf{psk})[2]$	$\mathsf{Hash}(H_3 \Vert \mathsf{InitHello.static})$
	5	-	-	$\mathsf{Hash}(H_4 \Vert \mathsf{InitHello.time})$
	6	$KDF_1(C_4,\ \mathsf{E_r^c} \Vert \mathsf{ct_2})$	-	$\mathsf{Hash}(H_5 \Vert \mathsf{E_r^c} \Vert \mathsf{ct_2})$
	7	$KDF_1(C_6,\ \mathsf{dh_{e_ie_r}} \Vert \mathsf{shk_2})$	-	-
	8	$KDF_1(C_7,\ \mathsf{dh_{s_ie_r}} \Vert \mathsf{shk_3})$	-	-
	9	$KDF_3(C_8,\ \mathsf{psk})[1]$	$KDF_3(C_8,\ \mathsf{psk})[3]$	$\mathsf{Hash}(H_6 \Vert KDF_3(C_8,\ \mathsf{psk})[2])$

Table 5.14: Key derivations. Steps 1 and 9 are common to all constructions. Session keys are computed as $(tk_i, tk_r) = KDF_2(C_9, \varnothing)$. Colored instructions are specific to WireGuard [Don17] (blue), to PQ-WireGuard [HNS+21] (green) and to PQ-WireGuard⋆ (orange). Other notations are from Table 5.1. C_k and H_k are chaining and hash values, κ_k are symmetric keys.

Chapter 6

Conclusion

> If you take a model's output as gospel, you're missing the point. Models are hypotheses, not truth machines.
>
> *Jessica Tierney*

In this thesis, we have addressed several critical aspects of modeling and verifying cryptographic protocols using automated tools. Automation of analysis has been a solution to the complexity of models and the errors that may arise from manual proofs. Unfortunately, the problem is not fully resolved, and modeling remains a highly challenging and complex task, even with automated tools.

Summary

Throughout our work, we have not only modeled and analyzed security protocols, but for each analyzed protocol where a symbolic model existed, we have provided detailed explanations for divergent results (when results differed). These findings may serve both as instructional examples and as documentation of modeling errors to avoid. In all proposed models, we have targeted specific aspects of cryptographic protocol verification using automated tools: the formal modeling of the protocol itself, the equational theory, the adversary model, the security properties, and the used automated verification tool. Our contributions are as follows:

A More Precise Modeling for Mix-Nets. This work introduces a more precise model for exponentiation and re-encryption Mix-Nets that, unlike prior abstract models, captures the exponentiation details. Through four case studies, we demonstrate the model's effectiveness in uncovering known and new attacks using ProVerif, mitigated only by strong zero-knowledge proofs (ZKPs). We propose two ZKP models (weak and strong), proving automatically that weak ZKPs fail to prevent those attacks. Additionally, we presented a refined ElGamal encryption model (independent of Mix-Nets), modeling keys explicitly as the results of exponentiation. This modeling revealed a subtle instantiation of an an attack againt exponentiation Mix-Net exploiting basis modification within the ElGamal encryption.

A Formal Analysis of WireGuard. In this work, we analyzed the WireGuard protocol in the symbolic model using Sapic$^+$, refining prior work with an enhanced adversary model. By incorporating numeros atomic capabilities (e.g., the compromise of the static/ephemeral/pre-shared keys, the compromise of the ECDH pre-computation, and the compromise of public key distribution), we rediscovered a known anonymity attack, uncovered a new vulnerability regarding the ECDH pre-computation used in WireGuard's implementation to optimize the computations, and proposed countermeasures to mitigate these weaknesses.

Hybridization of WireGuard. In this work, we analyzed the PQ-WireGuard protocol using ProVerif, DeepSec, and Tamarin. We found that it does not guarantee anonymity and exhibit Unkown Key-Share (UKS) attacks claimed to be infeasible.. This led us to a fixed version of the PQ-WireGuard, namely PQ-WireGuard*, which preserves all original security properties (agreement, key secrecy, mutual secrecy, forward secrecy, session uniqueness) while fixing the identified flaws. Building on WireGuard and PQ-WireGuard*, we designed Hybrid-WireGuard, combining classical and post-quantum security. Formal verification proved that any attack on Hybrid-WireGuard requires simultaneous breaches of both underlying protocols' securities, achieving our hybridization goal.

Limitation and Direction for Future Work

Our proposed future work includes applying our ElGamal and ZKPs models to additional protocols that use these cryptographic primitives. All existing ProVerif models using these primitives could potentially incorporate our refined equations. Current verification tools face limitations in finding the exact same algebraic attacks we described. While Tamarin provides the most comprehensive built-in model for Diffie-Hellman exponentiation (representing exponentiation as an associative-commutative function symbol), it does not allow exponentiation operators within user-defined equations. Although we used ProVerif for our models, several constraints persist in them. The current equational theories use a fixed generator, which restricts the number of possible exponentiations - a limitation particularly problematic as certain attacks require more than three exponentiation. Finally, our current analysis does not account for scenarios involving corrupted Mix-Nets. Proper modeling of one or more compromised

Mix-Nets would require incorporating additional exponentiations in the equational theory.

An additional research direction involves determining the minimal binding properties required for the KEMs in the PQ-WireGuard and Hybrid-WireGuard protocols to prevent re-encapsulation attacks. The Tamarin library proposed in [CDM24] can automate the analysis of these necessary KEM properties. We did not compare the results of our analysis with Rosenpass [VLZ$^+$24], which is an implementation of PQ-WireGuard generating a post-quantum shared secret usable as a WireGuard pre-shared key. Specifically, existing analyses do not permit direct evaluation of hybridization. Our framework's application to Rosenpass would allow rigorous protocol comparison and exact determination of its hybridization guarantees.

Our current protocol models for WireGuard and PQ-WireGuard present several limitations. Both WireGuard and PQ-WireGuard are stateful protocols: the Initiator generates a timestamp and includes it in the InitHello message, and the Responder only accepts the InitHello messages with strictly increasing timestamps, and stores and update them accordingly to prevent replay attacks. A symbolic modeling and analysis for both protocols as stateful ones would be necessary for comprehensive protocol analysis. Additionally, our models currently exclude timestamp compromise scenarios. Incorporating both timestamp compromise and stateful protocol modeling would enable detection of state disruption attacks [VLZ$^+$24].

We identified an attack against WireGuard's unlinkability. This attack requires modeling the protocol with states to be detected automatically. Additionally, we aim to investigate the field of unlinkability by proposing alternative formal definitions, as current definitions based on observational equivalence cannot be tested directly with automated tools. An investigation in this direction seems also promising.

We would also like to apply our methodology for searching for minimal offensive and defensive models to other protocols and compare them. For example, TLS appears to be a good candidate, and our approach could even build upon existing TLS models. It would also be interesting to optimize our search algorithms, which would streamline analysis and advance research in this direction.

Bibliography

[AAA$^+$22] Gorjan Alagic, Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, et al. Status report on the third round of the nist post-quantum cryptography standardization process. *US Department of Commerce, National Institute of Standards and Technology*, 2022.

[ABD$^+$19] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber algorithm specifications and supporting documentation. *NIST PQC Round*, 2(4):1–43, 2019.

[ABF17] Martín Abadi, Bruno Blanchet, and Cédric Fournet. The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication. *Journal of the ACM (JACM)*, 65(1):1 – 103, October 2017.

[Adi08] Ben Adida. Helios: web-based open-audit voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, page 335–348, USA, 2008. USENIX Association.

[AKTZ17] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. MCMix: Anonymous messaging via secure multiparty computation. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1217–1234, Vancouver, BC, August 2017. USENIX Association.

[AMAD$^+$24] Carlos Aguilar-Melchor, Nicolas Aragon, Jean-Christophe Deneuville, Philippe Gaborit, Jérôme Lacan, and Gilles Zémor. Efficient error-correcting codes for the HQC post-quantum cryptosystem. *Designs, Codes and Cryptography*, 92(12):4511–4530, October 2024.

[AMW19] Jacob Appelbaum, Chloe Martindale, and Peter Wu. Tiny WireGuard tweak. In Johannes Buchmann, Abderrahmane Nitaj, and

Tajje eddine Rachidi, editors, *AFRICACRYPT 19: 11th International Conference on Cryptology in Africa*, volume 11627 of *Lecture Notes in Computer Science*, pages 3–20, Rabat, Morocco, July 9–11, 2019. Springer, Cham, Switzerland.

[ANJ$^+$24] Prashant Agrawal, Abhinav Nakarmi, Mahabir Prasad Jhanwar, Subodh Sharma, and Subhashis Banerjee. Traceable mixnets. *Proceedings on Privacy Enhancing Technologies*, 2024(2):235–275, April 2024.

[ANS22] ANSSI. ANSSI views on the Post-Quantum Cryptography transition, 2022. https://cyber.gouv.fr/en/publications/anssi-views-post-quantum-cryptography-transition.

[ANWW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: Simpler, smaller, fast as MD5. In Michael J. Jacobson, Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 119–135, Banff, AB, Canada, June 25–28, 2013. Springer Berlin Heidelberg, Germany.

[AW07] Ben Adida and Douglas Wikström. Offline/online mixing. Cryptology ePrint Archive, Paper 2007/143, 2007.

[BAN90] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, February 1990.

[BBD$^+$20] Andrea Basso, Jose Maria Bermudo, Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Michiel Van Beirendonck, and Frederik Vercauteren. Saber: Mod-lwr based kem. Round-3 submission to the NIST PQC project,, 2020. https://www.esat.kuleuven.be/cosic/pqcrypto/saber/resources.html.

[BBK17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *IEEE Symposium on Security and Privacy (S&P'17)*, pages 483–503, San Jose, CA, May 2017. IEEE. **Distinguished paper award**.

[BBMP24] Sevdenur Baloglu, Sergiu Bursuc, Sjouke Mauw, and Jun Pang. Formal verification and solutions for estonian e-voting. In *Proceedings of the 19th ACM Asia Conference on Computer and Commu-*

nications Security, ASIA CCS '24, page 728–741, New York, NY, USA, 2024. Association for Computing Machinery.

[BBU13] Michael Backes, Fabian Bendun, and Dominique Unruh. Computational soundness of symbolic zero-knowledge proofs: Weaker assumptions and mechanized verification. In David Basin and John C. Mitchell, editors, *Principles of Security and Trust*, pages 206–225, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BC14] David Basin and Cas Cremers. Know your enemy: Compromising adversaries in protocol analysis. *ACM Trans. Inf. Syst. Secur.*, 17(2), November 2014.

[BCDS17] David Basin, Cas Cremers, Jannik Dreier, and Ralf Sasse. Symbolically analyzing security protocols using Tamarin. *ACM SIGLOG News*, 4(4):19–30, November 2017.

[BCG^{+}18] David Bernhard, Véronique Cortier, Pierrick Gaudry, Mathieu Turuani, and Bogdan Warinschi. Verifiability analysis of CHVote. Cryptology ePrint Archive, Paper 2018/1052, 2018.

[BCW22] Karthikeyan Bhargavan, Vincent Cheval, and Christopher Wood. A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello. In *CCS '22: 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 365–379, Los Angeles CA, United States, November 2022. ACM.

[BDD23] David Baelde, Alexandre Debant, and Stéphanie Delaune. Proving Unlinkability Using ProVerif Through Desynchronised Bi-Processes. In *2023 IEEE 36th Computer Security Foundations Symposium (CSF)*, pages 75–90, 2023.

[BDM20] David Baelde, Stéphanie Delaune, and Solène Moreau. A Method for Proving Unlinkability of Stateful Protocols. In *33rd IEEE Computer Security Foundations Symposium*, 33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020, Boston, United States, June 2020.

[BDP15] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, and Alfredo Pironti. Verified contributive channel bindings for compound authentication. In *ISOC Network and Distributed System Security Symposium – NDSS 2015*, San Diego, CA, USA, February 8–11, 2015. The Internet Society.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45, Santa Barbara, CA, USA, August 23–27, 1998. Springer Berlin Heidelberg, Germany.

[Bel20] Formal analysis of the belenios vs protocol. `https://members.loria.fr/AFilipiak/formal-analysis-of-the-belenios-vs-protocol/formal-analysis-of-the-belenios-vs-protocol-vote-confidentiality/formal-analysis-of-the-belenios-vs-protocol-vote-confidentiality-compromised-voting-server-authentication-credentials-and-voting-device/`, 2020.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228, New York, NY, USA, April 24–26, 2006. Springer Berlin Heidelberg, Germany.

[BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 263–280, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[BG19] Colin Boyd and Kai Gellert. A modern view on forward security. Cryptology ePrint Archive, Paper 2019/1362, 2019.

[BGZBH11] Gilles Barthe, Benjamin Grégoire, Santiago Zanella-Béguelin, and Sylvain Heraud. Computer-Aided Security Proofs for the Working Cryptographer. In *Advances in Cryptology - {CRYPTO} 2011 - 31st Annual Cryptology Conference*, Santa Barbara, United States, 2011.

[BHM20] Xavier Boyen, Thomas Haines, and Johannes Mueller. A verifiable and practical lattice-based decryption mix net with external auditing. Cryptology ePrint Archive, Paper 2020/115, 2020.

[BJKS24] Karthikeyan Bhargavan, Charlie Jacomme, Franziskus Kiefer, and Rolfe Schmidt. Formal verification of the PQXDH post-quantum key agreement protocol for end-to-end secure messaging. In Davide

Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.

[Bla09] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *Dependable and Secure Computing, IEEE Transactions on*, 5:193 – 207, 01 2009.

[Bla14] Bruno Blanchet. Automatic Verification of Security Protocols in the Symbolic Model: The Verifier ProVerif. In Alessandro Aldini, Javier Lopez, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII*, volume 8604 of *Lecture Notes in Computer Science*, pages 54–87. Springer, 2014.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12, Santa Barbara, CA, USA, August 23–27, 1998. Springer Berlin Heidelberg, Germany.

[BN05] Sébastien Briais and Uwe Nestmann. A formal semantics for protocol narrations. In Rocco De Nicola and Davide Sangiorgi, editors, *Trustworthy Global Computing*, pages 163–181, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. pages 626–643, 12 2012.

[BS16] Bruno Blanchet and Ben Smyth. Automated reasoning for equivalences in the applied pi calculus with barriers. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 310–324, 2016.

[CCD23a] Vincent Cheval, Véronique Cortier, and Alexandre Debant. Election Verifiability with ProVerif. In *CSF 2023 - 36th IEEE Computer Security Foundations Symposium*, Dubrovnik, Croatia, July 2023.

[CCD$^+$23b] Vincent Cheval, Cas Cremers, Alexander Dax, Lucca Hirschi, Charlie Jacomme, and Steve Kremer. Hash gone bad: Automated discovery of protocol attacks that exploit hash function weaknesses. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim,* *CA, USA, August 9-11, 2023*, pages 5899–5916. USENIX Association, 2023.

[CCK22] Vincent Cheval, Raphaëlle Crubillé, and Steve Kremer. Symbolic protocol verification with dice: process equivalences in the presence of probabilities. In *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*, pages 319–334, 2022.

[CD09] Véronique Cortier and Stéphanie Delaune. A method for proving observational equivalence. In *2009 22nd IEEE Computer Security Foundations Symposium*, pages 266–276, 2009.

[CDJZ23a] Cas Cremers, Alexander Dax, Charlie Jacomme, and Mang Zhao. Automated Analysis of Protocols that use Authenticated Encryption: Analysing the Impact of the Subtle Differences between AEADs on Protocol Security. In *USENIX Security 2023*, Anaheim, United States, August 2023. USENIX.

[CDJZ23b] Cas Cremers, Alexander Dax, Charlie Jacomme, and Mang Zhao. Automated analysis of protocols that use authenticated encryption: How subtle AEAD differences can impact protocol security. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 5935–5952, Anaheim, CA, USA, August 9–11, 2023. USENIX Association.

[CDM24] Cas Cremers, Alexander Dax, and Niklas Medinger. Keeping up with the kems: Stronger security notions for kems and automated analysis of kem-based protocols. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 1046–1060, New York, NY, USA, 2024. Association for Computing Machinery.

[CGG22] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. Features and usage of Belenios in 2022. In *The International Conference for Electronic Voting (E-Vote-ID 2022)*, Bregenz / Hybrid, Austria, October 2022.

[CGT17] Véronique Cortier, David Galindo, and Mathieu Turuani. A formal analysis of the Neuchâtel e-voting protocol. Research report, Inria Nancy - Grand Est, October 2017.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, feb 1981.

[CHSW22] Sofía Celi, Jonathan Hoyland, Douglas Stebila, and Thom Wiggers. A tale of two models: Formal verification of KEMTLS via Tamarin. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022: 27th European Symposium on Research in Computer Security, Part III*, volume 13556 of *Lecture Notes in Computer Science*, pages 63–83, Copenhagen, Denmark, September 26–30, 2022. Springer, Cham, Switzerland.

[CJ19a] Cas Cremers and Dennis Jackson. Prime, Order Please! Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman . In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 78–7815, Los Alamitos, CA, USA, June 2019. IEEE Computer Society.

[CJ19b] Cas Cremers and Dennis Jackson. Prime, order please! revisiting small subgroup and invalid curve attacks on protocols using diffie-hellman. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 78–7815, 2019.

[CJKK22] Vincent Cheval, Charlie Jacomme, Steve Kremer, and Robert Künnemann. SAPIC+: protocol verifiers of the world, unite! In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 3935–3952, Boston, MA, USA, August 10–12, 2022. USENIX Association.

[CKR18] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. The DEEPSEC prover. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, volume 10982 of *Lecture Notes in Computer Science*, pages 28–36. Springer, 2018.

[CKR24] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. Deepsec: Deciding equivalence properties for security protocols – improved theory and practice. *TheoretiCS*, Volume 3, March 2024.

[CKW09] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems. Research Report RR-6912, INRIA, 2009.

[CMR23] Vincent Cheval, José Moreira, and Mark Ryan. Automatic verification of transparency protocols. In *8th IEEE European Symposium* *on Security and Privacy, EuroS&P 2023, Delft, Netherlands, July 3-7, 2023.* IEEE, 2023.

[CPS19] Eric Crockett, Christian Paquin, and Douglas Stebila. Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. Cryptology ePrint Archive, Report 2019/858, 2019.

[CRST15a] Chris Culnane, Peter Y. A. Ryan, Steve Schneider, and Vanessa Teague. vvote: A verifiable voting system. *ACM Trans. Inf. Syst. Secur.*, 18(1), June 2015.

[CRST15b] Chris Culnane, Peter Y. A. Ryan, Steve Schneider, and Vanessa Teague. Vvote: A verifiable voting system. *ACM Trans. Inf. Syst. Secur.*, 18(1), jun 2015.

[CS11] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21:297–311, 06 2011.

[CT07] Liqun Chen and Qiang Tang. Bilateral unknown key-share attacks in key agreement protocols. *IACR Cryptology ePrint Archive*, 2007:209, 01 2007.

[Dav82] G. Davida. Chosen signature cryptanalysis of the RSA (MIT) public key cryptosystem. *TR-CS-82-2, Dept. of Electrical Engineering and Computer Science, Univ. of Wisconsin, Milwaukee, Wisconsin*, 1982.

[DGK$^+$14] Jannik Dreier, Rosario Giustolisi, Ali Kassem, Pascal Lafourcade, Gabriele Lenzini, and Peter Y. A. Ryan. Formal analysis of electronic exams. In *2014 11th International Conference on Security and Cryptography (SECRYPT)*, pages 1–12, 2014.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DJL14] Jannik Dreier, Hugo Jonker, and Pascal Lafourcade. Secure Auctions Without Cryptography (extended version). Technical report, ETH Zurich, April 2014.

[DKR06] S. Delaune, S. Kremer, and M. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, pages 12 pp.–42, 2006.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *J. Comput. Secur.*, 17(4):435–487, dec 2009.

[DLM24] Jannik Dreier, Pascal Lafourcade, and Dhekra Mahmoud. Shaken, not stirred - automated discovery of subtle attacks on protocols using Mix-Nets. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 3135–3150, Philadelphia, PA, August 2024. USENIX Association.

[DLMS99] Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. Undecidability of bounded security protocols. 1999.

[DM17] Jason A Donenfeld and Kevin Milner. Formal verification of the WireGuard protocol. *Technical Report, Tech. Rep.*, 2017.

[DM18] Jason A. Donenfeld and Kevin Milner. Formal verification of the wireguard protocol. `https://www.wireguard.com/papers/wireguard-formal-verification.pdf`, 2018.

[DMWG23] Quang Dao, Jim Miller, Opal Wright, and Paul Grubbs. Weak fiat-shamir attacks on modern proof systems. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 199–216. IEEE, 2023.

[Don17] Jason A. Donenfeld. WireGuard: Next generation kernel network tunnel. In *ISOC Network and Distributed System Security Symposium – NDSS 2017*, San Diego, CA, USA, February 26 – March 1, 2017. The Internet Society.

[Don20] Jason A. Donenfeld. Technical whitepaper of wireguard. `https://www.wireguard.com/papers/wireguard.pdf`, 2020.

[Don23] Jason A. Donenfeld. Go implementation of wireguard. `https://git.zx2c4.com/wireguard-go/about/`, 2023.

[Don24] Jason A Donenfeld. WireGuard known limitations. `https://www.wireguard.com/known-limitations/`, 2024.

[DP18] Benjamin Dowling and Kenneth G. Paterson. A cryptographic analysis of the WireGuard protocol. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18: 16th International Conference on Applied Cryptography and Network Security*, volume 10892 of *Lecture Notes in Computer Science*, pages 3–21, Leuven, Belgium, July 2–4, 2018. Springer, Cham, Switzerland.

[Dre13] Jannik Dreier. *Formal Verification of Voting and Auction Protocols : From Privacy to Fairness and Verifiability.* PhD thesis, 2013. Thèse de doctorat dirigée par Lakhnech, Yassine et Lafourcade, Pascal Informatique Grenoble 2013.

[DRS20] Benjamin Dowling, Paul Rösler, and Jörg Schwenk. Flexible authenticated and confidential channel establishment (fACCE): Analyzing the noise protocol framework. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12110 of *Lecture Notes in Computer Science*, pages 341–373, Edinburgh, UK, May 4–7, 2020. Springer, Cham, Switzerland.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18, Santa Barbara, CA, USA, August 19–23, 1984. Springer Berlin Heidelberg, Germany.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Eur20] European Telecommunications Standards Institute (ETSI). ETSI TS 103 744 V1.1.1 (2020-12): Quantum-safe Hybrid Key Exchanges, 2020. `https://www.etsi.org/deliver/etsi_ts/103700_103799/103744/01.01.01_60/ts_103744v010101p.pdf`.

[Fed21] Federal Office for Information Security (BSI). Quantum Technologies and Quantum-Safe Cryptography, 2021. `https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Quantentechnologien-und-Post-Quanten-Kryptografie/quantentechnologien-und-post-quanten-kryptografie_node.html`.

[FK11] Sheila Frankel and Suresh Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, RFC Editor, February 2011.

[FMJ24] Markus Friedl, Jan Mojzis, and Simon Josefsson. Secure Shell (SSH) Key Exchange Method Using Hybrid Streamlined NTRU

Prime sntrup761 and X25519 with SHA-512: sntrup761x25519-sha512. Internet-Draft draft-josefsson-ntruprime-ssh-03, IETF Secretariat, August 2024.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings on Advances in Cryptology—CRYPTO '86*, page 186–194, Berlin, Heidelberg, 1987. Springer-Verlag.

[FSXY12] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. Strongly secure authenticated key exchange from factoring, codes, and lattices. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public Key Cryptography – PKC 2012*, pages 467–484, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[GGCG$^+$21] Stefan-Lukas Gazdag, Sophia Grundner-Culemann, Tobias Guggemos, Tobias Heider, and Daniel Loebenberger. A formal analysis of ikev2's post-quantum extension. In *Proceedings of the 37th Annual Computer Security Applications Conference*, ACSAC '21, page 91–105, New York, NY, USA, 2021. Association for Computing Machinery.

[GHS$^+$20] Guillaume Girol, Lucca Hirschi, Ralf Sasse, Dennis Jackson, Cas Cremers, and David Basin. A Spectral Analysis of Noise: A Comprehensive, Automated, Formal Analysis of Diffie-Hellman Protocols. In *USENIX 2020 - 29th Usenix Security Symposium*, Virtual, United States, August 2020.

[GJJS04] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. Universal re-encryption for mixnets. In Tatsuaki Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, pages 163–178, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Gjø12] Kristian Gjøsteen. The norwegian internet voting protocol. In Aggelos Kiayias and Helger Lipmaa, editors, *E-Voting and Identity*, pages 1–18, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[GLR14] Rosario Giustolisi, Gabriele Lenzini, and Peter Y. A. Ryan. Remark!: A secure protocol for remote exams. In *Security Protocols Workshop*, 2014.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GMP21] Paul Grubbs, Varun Maram, and Kenneth G. Paterson. Anonymous, robust post-quantum public key encryption. Cryptology ePrint Archive, Paper 2021/708, 2021.

[Gol96] D. Gollmann. What do we mean by entity authentication? In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, SP '96, page 46, USA, 1996. IEEE Computer Society.

[HBD17] Lucca Hirschi, David Baelde, and Stéphanie Delaune. A method for unbounded verification of privacy-type properties. *CoRR*, abs/1710.02049, 2017.

[HGS21] Thomas Haines, Rajeev Goré, and Bhavesh Sharma. Did you mix me? formally verifying verifiable mix nets in electronic voting. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1748–1765. IEEE, 2021.

[HKL23] Rolf Haenni, Reto Koenig, and Philipp Locher. Private internet voting on untrusted voting devices. In *7th Workshop on Advances in Secure Electronic Voting*. 2023.

[HKLD17] Rolf Haenni, Reto E. Koenig, Philipp Locher, and Eric Dubuis. CHVote protocol specification. Cryptology ePrint Archive, Paper 2017/325, 2017.

[HM20] Ross Horne and Sjouke Mauw. Discovering epassport vulnerabilities using bisimilarity. *Logical Methods in Computer Science*, Volume 17, Issue 2, 02 2020.

[HMA02] Specifications for the keyed-hash message authentication code. National Institute of Standards and Technology (NIST), FIPS PUB 198, U.S. Department of Commerce, March 2002.

[HMLP25] Ayoub Ben Hassen, Dhekra Mahmoud, Pascal Lafourcade, and Maxime Puys. Formal Analysis of SDNsec: Attacks and Corrections for Payload, Route Integrity and Accountability. In *ASIA CCS '24: Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, Hanoi, Vietnam, August 2025.

[HNS$^+$20] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Fiona Johanna Weber, and Philip R. Zimmermann. Post-quantum wireguard. `https://github.com/KPN-CISO/pq-wg-theory/blob/master/pq_wireguard.spthy`, 2020. Accessed: 2025-04-6.

[HNS^{+}21] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Fiona Johanna Weber, and Philip R. Zimmermann. Post-quantum WireGuard. In *2021 IEEE Symposium on Security and Privacy*, pages 304–321, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[HS11] Rolf Haenni and Oliver Spycher. Secure internet voting on limited devices with anonymized DSA public keys. In *2011 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 11)*, San Francisco, CA, August 2011. USENIX Association.

[ISO09] Common criteria for information technology security evaluation - part 2: Security functional components. Standard, International Organization for Standardization, July 2009.

[IVX23] Specification of IVXV estonian voting protocols, 2023. `https://www.valimised.ee/sites/default/files/2023-02/IVXV-protocols.pdf`.

[Jak98] Markus Jakobsson. A practical mix. In *International Conference on the Theory and Application of Cryptographic Techniques*, 1998.

[JCCS19] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. Seems legit: Automated analysis of subtle attacks on protocols that use signatures. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2165–2180. ACM, 2019.

[JJ01] Markus Jakobsson and Ari Juels. An optimally robust hybrid mix network. pages 284–292, 08 2001.

[JKKR23] Charlie Jacomme, Elise Klein, Steve Kremer, and Maïwenn Racouchot. A comprehensive, formal and automated analysis of the EDHOC protocol. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 5881–5898, Anaheim, CA, USA, August 9–11, 2023. USENIX Association.

[KBB17] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *2017*

IEEE European Symposium on Security and Privacy, pages 435–450, Paris, France, April 26–28, 2017. IEEE Computer Society Press.

[KHNE10] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996, RFC Editor, September 2010.

[KMST16] Ralf Kuesters, Johannes Mueller, Enrico Scapin, and Tomasz Truderung. sElect: A lightweight verifiable remote voting system. Cryptology ePrint Archive, Paper 2016/438, 2016.

[KMW12] Shahram Khazaei, Tal Moran, and Douglas Wikström. A mix-net from any cca2 secure cryptosystem. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, pages 607–625, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[KNB18] Nadim Kobeissi, Georgio Nicolas, and Karthikeyan Bhargavan. Noise explorer: Fully automated modeling and verification for arbitrary noise protocols. Cryptology ePrint Archive, Paper 2018/766, 2018.

[KNB19] Nadim Kobeissi, Georgio Nicolas, and Karthikeyan Bhargavan. Noise explorer: Fully automated modeling and verification for arbitrary noise protocols. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 356–370. IEEE, 2019.

[KNW03] Deepak Kapur, Paliath Narendran, and Lida Wang. An e-unification algorithm for analyzing protocols that use modular exponentiation. In Robert Nieuwenhuis, editor, *Rewriting Techniques and Applications*, pages 165–179, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Kra05] Hugo Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, pages 546–566, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648, Santa Barbara, CA, USA, August 15–19, 2010. Springer Berlin Heidelberg, Germany.

[KS23] Ehren Kret and Rolfe Schmidt. The PQXDH key agreement protocol, 2023.

[KSH24] Panos Kampanakis, Douglas Stebila, and Torben Hansen. PQ/T Hybrid Key Exchange in SSH. Internet-Draft draft-kampanakis-curdle-ssh-pq-ke-03, IETF Secretariat, August 2024.

[KTV14] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Formal analysis of chaumian mix nets with randomized partial checking. In *2014 IEEE Symposium on Security and Privacy*, pages 343–358, 2014.

[LBB19] Benjamin Lipp, Bruno Blanchet, and Karthikeyan Bhargavan. A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 231–246, 2019.

[LHT16] Adam Langley, Mike Hamburg, and Sean Turner. Elliptic Curves for Security. RFC 7748, RFC Editor, January 2016.

[lib] liboqs. `https://github.com/open-quantum-safe/liboqs`.

[Lin20] Linux 5.6. `https://kernelnewbies.org/Linux_5.6`, 2020.

[LMMOA24] Pascal Lafourcade, Dhekra Mahmoud, Gael Marcadet, and Charles Olivier-Anclin. Transferable, Auditable and Anonymous Ticketing Protocol. In *2024 Asia Conference on Information, Computer and Communications Security*, Singapore, Singapore, July 2024.

[LMR24] Pascal Lafourcade, Dhekra Mahmoud, and Sylvain Ruhault. A Unified Symbolic Analysis of WireGuard. In *Usenix Network and Distributed System Security Symposium*, San Diego (CA), United States, February 2024.

[LMRT25] Pascal Lafourcade, Dhekra Mahmoud, Sylvain Ruhault, and Abdulrahman Taleb. Sapic files for pq-wireguard and hybrid-wireguard. `https://osf.io/rdy7c/?view_only=8833688721584887b5d84698383bf0d7`, 2025. Accessed: 2025-04-11.

[LN15] Adam Langley and Yoav Nir. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539, RFC Editor, May 2015.

[Low95] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[Low97] G. Lowe. A hierarchy of authentication specifications. In *Proceedings 10th Computer Security Foundations Workshop*, pages 31–43, 1997.

[LSB24] Felix Linker, Ralf Sasse, and David Basin. A Formal Analysis of Apple's iMessage PQ3 Protocol. *Cryptology ePrint Archive*, 2024.

[Lud23] Justin Ludwig. Wireguard key on an openpgp card. `https://www.procustodibus.com/blog/2023/03/openpgpcard-wireguard-guide/`, 2023.

[Mil89] R. Milner. *Communication and Concurrency.* Ph/AMA Series in Marketing. Prentice Hall, 1989.

[MLK24] Module-lattice-based key-encapsulation mechanism standard. National Institute of Standards and Technology, NIST FIPS PUB 203, U.S. Department of Commerce, August 2024.

[MLM24] Mounira Msahli, Pascal Lafourcade, and Dhekra Mahmoud. Formal Analysis of C-ITS PKI protocols. In *SECRYPT 2024 : International Conference on Information Security and Cryptography*, Dijon, France, July 2024.

[Möd12] Sebastian Mödersheim. Diffie-hellman without difficulty. In Gilles Barthe, Anupam Datta, and Sandro Etalle, editors, *Formal Aspects of Security and Trust*, pages 214–229, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Mül22] Johannes Müller. Breaking and fixing vote privacy of the estonian e-voting protocol ivxv. In *Financial Cryptography Workshops*, 2022.

[MW04] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In Moni Naor, editor, *Theory of Cryptography*, pages 133–151, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Nat23] National Institute of Standards and Technology (NIST). NIST SP 1800-38: Migration to Post-Quantum Cryptography: Preparation for Considering the Implementation and Adoption of Quantum Safe Cryptography, 2023. `https://csrc.nist.gov/pubs/sp/1800/38/iprd-(1)`.

[Nor] NordVPN. NordLynx. `https://nordvpn.com/blog/nordlynx-protocol-wireguard/`.

[NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the Association for Computing Machinery*, 21(21):993–999, December 1978.

[Ope01] OpenVPN. `https://openvpn.net/`, 2001.

[oSN25] National Institute of Standards and Technology NIST. Nist selects hqc as fifth algorithm for post-quantum encryption. `https://www.nist.gov/news-events/news/2025/03/nist-selects-hqc-fifth-algorithm-post-quantum-encryption`, 2025. Accessed: 2025-04-3.

[Per18] Trevor Perrin. The Noise protocol framework. *noiseprotocol, Protocol Revision*, 34, 2018.

[Pfi94] Birgit Pfitzmann. Breaking efficient anonymous channel. In *International Conference on the Theory and Application of Cryptographic Techniques*, 1994.

[PIK93] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer, 1993.

[Pos80] J. Postel. User Datagram Protocol. RFC 768, RFC Editor, August 1980.

[PP90] Birgit Pfitzmann and Andreas Pfitzmann. How to break the direct RSA-implementation of mixes. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 373–381, Houthalen, Belgium, April 10–13, 1990. Springer Berlin Heidelberg, Germany.

[PQ01] O. Pereira and J.-J. Quisquater. A security analysis of the cliques protocols suites. In *Proceedings. 14th IEEE Computer Security Foundations Workshop, 2001.*, pages 73–81, 2001.

[pqs] pq-strongswan. `https://github.com/strongX509/docker/tree/master/pq-strongswan`.

[PST20] Christian Paquin, Douglas Stebila, and Goutam Tamvada. Benchmarking post-quantum cryptography in TLS. In Jintai Ding and

Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 72–91, Paris, France, April 15–17, 2020. Springer, Cham, Switzerland.

[RA12] Pance Ribarski and Ljupcho Antovski. Mixnets: Implementation and performance evaluation of decryption and re-encryption types. *Journal of Computing and Information Technology*, 20:225–231, 09 2012.

[RGL22] Mohammadamin Rakeei, Rosario Giustolisi, and Gabriele Lenzini. Secure internet exams despite coercion, 2022.

[Ros05] A. Roscoe. *The Theory and Practice of Concurrency.* 01 2005.

[RS11] Mark D. Ryan and Ben Smyth. Applied pi calculus. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, chapter 6. IOS Press, 2011.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, February 1978.

[Rya15] Peter Y. Ryan. Crypto santa. In *LNCS Essays on The New Codebreakers - Volume 9100*, page 543–549, Berlin, Heidelberg, 2015. Springer-Verlag.

[SA15] Markku-Juhani O. Saarinen and Jean-Philippe Aumasson. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). RFC 7693, RFC Editor, November 2015.

[SD18] A Suter-Döring. Formalizing and verifying the security protocols from the noise framework, bachelor thesis. `https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/noise_suter-doerig.pdf`, 2018.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press.

[SKD20] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Post-quantum authentication in TLS 1.3: A performance study. Cryptology ePrint Archive, Report 2020/071, 2020.

[SM16] Douglas Stebila and Michele Mosca. Post-quantum key exchange for the internet and the open quantum safe project. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016: 23rd Annual International Workshop on Selected Areas in Cryptography*, volume 10532 of *Lecture Notes in Computer Science*, pages 14–37, St. John's, NL, Canada, August 10–12, 2016. Springer, Cham, Switzerland.

[SP07] Krishna Sampigethaya and Radha Poovendran. A survey on mix networks and their secure applications. *Proceedings of the IEEE*, 94:2142 – 2181, 01 2007.

[SSH14] Efstathios Stathakidis, Steve Schneider, and James Heather. Robustness modelling and verification of a mix net protocol. pages 131–150, 12 2014.

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1461–1480, Virtual Event, USA, November 9–13, 2020. ACM Press.

[Ste24] Douglas Stebila. Security analysis of the iMessage PQ3 protocol. Cryptology ePrint Archive, Paper 2024/357, 2024.

[Swi22] Symbolic analysis of the swiss post voting system. `https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/master/Symbolic-models`, 2022.

[TTB$^+$23] CJ. Tjhai, M. Tomlinson, G. Bartlett, S. Fluhrer, D. Van Geest, O. Garcia-Morchon, and V. Smyslov. Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2). RFC 9370, RFC Editor, May 2023.

[TW10] Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In Daniel J. Bernstein and Tanja Lange, editors, *Progress in Cryptology – AFRICACRYPT 2010*, pages 100–113, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[VLZ$^+$24] Karolin Varner, Benjamin Lipp, Wanja Zaeke, Lisa Schmidt, and Prabhpreet Dua. Rosenpass. `https://rosenpass.eu/whitepaper.pdf`, 2024.

[WG06] Douglas Wikström and Jens Groth. An adaptively secure mix-net without erasures. In Michele Bugliesi, Bart Preneel, Vladimiro

Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 276–287, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Wik04a] Douglas Wikström. Five practical attacks for “optimistic mixing for exit-polls”. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, pages 160–174, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Wik04b] Douglas Wikström. A universally composable mix-net. In Moni Naor, editor, *Theory of Cryptography*, pages 317–335, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Wik05] Douglas Wikström. A sender verifiable mix-net and a new proof of a shuffle. In Bimal Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, pages 273–292, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[WSF$^+$03] Burkhart Wolff, Oliver S, Hannes Federrath, Stefan Kispsell, and Andreas Pfitzmann. Towards a Formal Analysis of a Mix Network. Technical report, Institut für Informatik Albert–Ludwigs–Universität Freiburg, 2003.