Universidade do Minho
Escola de Engenharia

Tiago Miguel Soares Fernandes

Evaluation of key-agreement protocols ba

Março de 2013

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Tiago Miguel Soares Fernandes

Evaluation of key-agreement protocols ba

Dissertação de Mestrado
Mestrado em Engenharia Informática ▼

Trabalho realizado sob orientação de
Professor Manuel Bernardo Barbosa

Março de 2013

Evaluation of key-agreement protocols based on weak secrets

Author: Tiago Miguel Soares Fernandes

Supervisor: Manuel Bernardo Barbosa

University of Minho

March, 2013

Acknowledgments

Firstly, I would like to thank my Supervisor, Prof. Manuel Bernardo Barbosa, for his counseling and support through this dissertation.

I wish to thank Tiago and Isabel, my dearest Father and Mother, for all the support through all these years.

I like to thank specially to Ana Palhares, when I needed strength she was always there for me. I would like to thank my good friends Jorge Miranda, Nuno Correia, João Sanches, Luís Mascarenhas, Rui Abreu, Mariana Fernandes, Cristina Oliveira, Carlène Barbosa, Pedro Fernandes, Mário Lameiras, Rita Anjo, Tiago Dias, Mário Araújo, Ana Gonçalves, Rui Fonseca, Rui Costa, Rui Gama, Joana Presa, Tiago Castro, Nádia Silva and Afonso Arriaga.

I acknowledge that my work was supported by the *Fundação para a Ciência e Tecnologia* (FCT) under the project "WITS – Wireless Information Theoretic Security - Refª PTDC/EIA/71362/2006"

Resumo

Duas entidades desejam comunicar de forma segura através de um canal inseguro na presença de um adversário. Para isso acordam uma chave criptográfica forte a partir de uma fonte fraca de aleatoriedade, extraída com base nas características físicas da rede onde comunicam. Nesta dissertação avaliamos as contrapartidas entre dois protocolos: um onde as garantias de segurança se baseiam em noções de teoria da informação — o protocolo *Authenticated Key Agreement (AKA)* proposto em [7] — e que foi especialmente criado para este cenário; e outro que baseia a sua segurança em argumentos computacionais — o protocolo *Password-Authenticated Key Exchange (PAKE)* proposto em [11]. Para este efeito efectuamos uma análise detalhada da segurança concreta de ambos os protocolos, considerando que em ambos os casos se pretende acordar uma nova chave de 128 bits.

Abstract

Two agents want to securely communicate on a insecure channel in the presence of an adversary. For that they agree in a strong cryptographic key based on a weak-source of randomness stemming from the physical network characteristics where these agents communicate. In this dissertation we evaluate the tradeoffs between two protocols: an information theoretically-secure Authenticated Key Agreement (AKA) [7] that was specifically designed for this scenario; and a Password-Authenticated Key Exchange (PAKE) protocol [11] whose security guarantees are based on computational arguments. To this end, we carry out an analysis of the concrete security of both protocols, considering in both cases that the goal is to agree on a fresh 128-bit secret key.

Contents

Chapter 1

Introduction

The topic for this dissertation was proposed in the context of the FCT-funded Wireless Information Theoretic Security (WITS) project, where the interaction between physical-layer security mechanisms and higher-layer security protocols for wireless networks is explored. Physical-layer security mechanisms take advantage of the characteristics of communications channels in order to provide a security functionality, which typically consists of the following:

- Two agents, Alice and Bob aim to securely transfer information across a communications channel.
- A passive adversary Eve is able to eavesdrop on the communications, according to what is known as the *wiretap channel* model.
- Taking advantage of the knowledge of the communications channel connecting them, and the fact that the quality of this channel will be (perhaps occasionally) better than that available to Eve, Alice is able to transfer to Bob a random string of which Eve has only partial knowledge.
- This random string, which we call a *weak secret* following [7], is usually modeled as being sampled from a random variable W shared between Alice and Bob.

Physical layer security results are typically formulated in the information-theoretical setting, and they describe the guarantees provided to Alice and Bob regarding Eve's uncertainty (or equivocation) with respect to the shared random variable W.

Given the security bootstrap provided by these physical security results, a higher level protocol is then applied to actually securely transfer data from Alice to Bob. This higher level protocol is typically carried out in two steps:

- A shared secret key k is derived from the weak secret.

- A cryptographic protocol is used to transfer data across the channel using k.

Assuming a passive adversary, the solutions for both these problems have been extensively studied. The typical solution is to use a *privacy amplification* protocol [13] to derive the secret key, and then a one-time-pad to convey the information across the channel. This solution provides an overall information-theoretical security level whereby no assumptions are made about the computational capabilities of the adversary. Nevertheless, it is clear that the one-time-pad can be replaced by alternative secure data transfer protocols, in order to improve performance or to protect against different types of adversaries. For example, one could simply use a computationally-secure authenticated encryption scheme to thwart active attacks by computationally bounded adversaries during the data transfer stage and securely reuse the shared secret key several times.

The natural question that arises is then: does a similar tradeoff between information-theoretic security and computational security make sense at other levels in the protocol stack? This question is partially addressed in this dissertation: we study the tradeoffs between information-theoretic and computational security solutions to deal with active attackers during the key derivation stage.

In cryptographic terminology, secure key derivation in the presence of active adversaries is the functionality provided by Authenticated Key Agreement (AKA) or Authenticated Key Exchange (AKE) protocols. In the WITS scenario, such AKA protocols must be able to derive a secure secret key from a weak secret. In the information-theoretical (IT) security setting, we analyze a concrete IT-AKA protocol [7] that was specifically designed for this scenario, and find that there are practical limitations to using such a protocol in the real-world. Indeed, even if efficient instantiations of all required components are available (or if one is willing to assume that some algorithm – e.g. a cryptographic hash function – provides the necessary functionality) one finds that the protocol puts a significant strain on the physical layer security mechanism: to derive a single 128-bit key at 128-bit security level, the protocol requires an input of a weak secret of almost 512 times that size. Furthermore, each weak secret can only be used to derive a single secret key.

In the computational security setting, the most natural counterpart to a weak secret is a password. Indeed, several Password-Authenticated Key Exchange (PAKE) solutions have been proposed which satisfy the intuitive security definition that an active attacker should only succeed in attacking the key exchange protocol if it successfully guesses the password. In particular, eavesdropping on a polynomial number of executions of the protocol (and even obtaining previously established secret keys), the adversary should not learn (in the computational sense) anything about the shared weak secret. Furthermore, the most efficient attack of an active adversary should be to test all possible weak secret values. We analyze the practical applicability of a concrete PAKE protocol [11] to the WITS scenario. We identify a typical application scenario for the PAKE protocol, whereby higher-level protocols will be fed with a fresh 128-bit secret key derived from the most recent weak secret released by the physical layer security mechanism. In this scenario, the security of the key can be seen as all-or-nothing: it is either the case that the adversary has actively established a rogue secret key by mounting an on-line attack and guessing the weak secret; or it is the case that the secret key is computationally hidden from the adversary at a 128-bit security level. We carry out a detailed analysis of the concrete computational security of the selected PAKE protocol in order to accurately estimate the values of all parameters that must be instantiated in its building blocks to obtain the desired security level. We evaluate the performance of the selected PAKE protocol at the desired security level by presenting an implementation of the selected PAKE protocol for the identified parametrisation and showing some benchmarking results.

Document structure

We start with Chapter 2 where we present security definitions, notions, and cryptographic primitives including their syntax, attack models, and security goals that will be used in the following chapters. In Chapter 3, we present the definition and security model of an information-theoretically secure Authenticated Key Agreement protocol, followed by a particular instance of such a protocol by from Dodis and Wichs [7]. We perform concrete security analysis of this protocol for 128 bit security, and present benchmarking results from a prototype implementation. Analogously, in Chapter 4, we describe the concept and security model for a computationally secure Password-Authenticated Key agreement protocol, and discuss a concrete instantiation proposed by Katz, Ostrovsky and Yung [11]. We also perform a concrete security analysis of this protocol for 128 bit security, and present benchmarking results from a prototype implementation. We conclude the paper with a discussion of our results in Chapter 5

Chapter 2

Preliminaries

2.1 Provable Security

Since the beginning of public-key cryptography, many suited algorithmic problems for cryptography have been proposed and many cryptographic schemes have been designed with more or less heuristics arguments of their security. If a cryptographic algorithm could cope cryptanalytic attacks for several years, it was often considered somehow as validation procedure. Several schemes take a long periods of time before being broken. Pointcheval [14] points out as an example the *Chor-Rivest* cryptosystem, that took more than 10 years to be totally broken. Before this attack was found, the scheme was believed to be strongly secure. This shows that the absence of attacks in some point in time must not be interpreted as security validation for some scheme. A different paradigm is given by the provable security concept. This is a line of research which tries to provide proofs in the framework of complexity theory. Commonly known as reductionist security proofs, these proofs provide reductions from a well-studied problem, such as RSA or the discrete logarithm, to an attack against some cryptographic protocol.

Computational Security

Many security goals in symmetric and asymmetric cryptography cannot be efficiently realized unless one relies on some computational assumption. There are two major families of computational assumptions in number theory-based public-key cryptography:

- the schemes based on integer factoring, and on the RSA problem. The most famous intractable problem is factorization of two integers: while it is easy to multiply two prime integers p and q to get the product $n = p.q$, it is not simple to decompose n into its prime factors p and q.

- the schemes based on the discrete logarithm problem, and on the Diffie-Hellman problems, in any “suitable” group.

Given such objects and thus computational assumptions about the intractability of the inversion without possible trapdoors, we would like that security could be achieved without extra assumptions. This fact is only formally proven when it is shown that an attacker against the cryptographic protocol can be used as sub-part in an algorithm that can break the computational assumption. Such arguments are called reductions. Reductions establish partial order between computational assumptions, also between intractable problems. As the required assumption is weaker, the more secure the cryptographic scheme is [14].

A reduction, in complexity theory, is an algorithm which uses an attacker as a sub-part in a global algorithm. If this reduction is efficient (i.e. the reduction is a polynomial-time algorithm), then the attack of the cryptographic protocol is at least as hard as inverting the function: if one has a polynomial algorithm to solve the latter problem, one can polynomially solve the former one. The associated asymptotic security definition typically requires that the success probability of any probabilistic, polynomial time algorithm be a negligible function of the security parameter. Sometimes hidden in security reductions based on these so-called asymptotic notions of security are very large factors, which imply that a successful attack on the scheme may not translate into a break of the computational assumption, except possibly for huge values of the security parameter (i.e. key sizes). Concrete security refers to a security reduction that does not rely asymptotic notions of security, rather manipulating concrete values for the parameters that specify the security level of the scheme, and the power of the adversary (namely its execution time), and using them to express an adversary’s probability of success.

Information-Theoretic security

An information-theoretically secure cryptosystem derives its security from arguments that establish an adversary’s probability of success independently of its computational power. This means essentially that computational assumptions have no place in an information-theoretical security proof, since they are not valid once one considers unbounded adversaries. Put simply, in an information-theoretically secure scheme, it is proven that the adversary does not have enough information to succeed in its attack. So, even if the adversary is computationally unbounded, encryption schemes based on information-theory can be proven secure [10].

A special case of information-theoretic schemes are encryption schemes displaying a property called perfect secrecy: the one-time pad is its most famous example. The main issue common to all schemes with perfect secrecy is that for any perfectly secret encryption scheme we must have a key space that is at least as large as the message space and can be used only once. If the key space consists of fixed-length keys, and the message space consists of all messages of some fixed length, this implies that the key must be as long as the message. Such practical limitations of information-theoretically secure schemes are common, and justify the pragmatic approach of considering computationally bounded adversaries. By making this relaxation and admitting a conceptually weaker security level, modern cryptography has been able to provide efficient solutions to very complex information security protection scenarios, that provide an adequate level of assurance for most practical applications.

2.2 Digital Signature Schemes

Digital Signatures schemes are a mechanism that protect the authenticity of a digital message or document. Digital signature schemes allow a signer who has established a public key pk to sign a message in such way that any other party who knows pk (and knows that the public key was established by the signer) can verify that the message originated from the signer and has not been modified in any way.

Syntax

A signature scheme is a tuple of three probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ satisfying the following:

- The key-generating algorithm Gen takes as input a security parameter 1^λ and outputs a pair of keys (pk, sk), public and private key respectively.

- The signing algorithm Sign takes as input a private key sk and a message $m \in \{0,1\}^*$. It outputs a signature σ, denoted as $\sigma \leftarrow \mathsf{Sign}(sk, m)$.

- The deterministic verification algorithm Vrfy takes as input a public key pk, a message m, and a signature σ. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ invalid signature. $b := \mathsf{Vrfy}(pk, m, \sigma)$.

Correctness

For every λ, every (pk, sk) outputted by $\mathsf{Gen}(1^\lambda)$ and every $m \in \{0,1\}^{l(\lambda)}$, it holds that $\mathsf{Vrfy}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

If $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is such that for every (pk, sk) output by $\mathsf{Gen}(1^\lambda)$, algorithm Sign is only defined for messages $m \in \{0,1\}^{l(\lambda)}$ (and Vrfy outputs 0 for $m \notin \{0,1\}^{l(\lambda)}$), then $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is a signature scheme for messages of length $l(\lambda)$ [4].

Security model

Considering a signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$. We consider the notion presented in [4] of strong existential unforgeability under an adaptive chosen-message attack. This can be described as an attack game as follows.

- The challenger runs Gen. It gives the adversary the resulting public key pk and keeps the private key sk to itself.
- The adversary issues signature queries $m'_1, \ldots, m'_q$, denoted as $\mathcal{P} = \{m'_1, \ldots, m'_q\}$, with q as the maximum number of queries allowed. The challenger responds by running Sign to generate signatures σ'_i of m'_i and sending σ'_i to the adversary. The set of issued signatures is denoted as $\mathcal{Q} = \{\sigma'_1, \ldots, \sigma'_q\}$.
- Finally the adversary outputs a pair (m, σ). The adversary wins if σ is a valid signature of m according to Vrfy and $\sigma \notin \mathcal{Q}$.

The advantage of an adversary $\mathcal{A}$ in attacking the signature scheme is defined as the probability that $\mathcal{A}$ wins the game described.

Definition 1. *([4]) A signature scheme* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is strongly existentially unforgeable under an adaptive chosen-message attack, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ wins the game above (i.e. its advantage) is negligible in the security parameter.*

In [4] a weaker version of unforgeability was also considered, properly called weak existential unforgeability. In this case, the adversary must only use a message m that was not previously queried to the Sign oracle.

We also use a weaker definition of digital signatures denoted one-time signatures. This type of signature is secure as long it is used to sign at most one message. In this particular case of signatures schemes, its security model only allows the adversary to submit one query $(q = 1)$ to Sign oracle.

2.3 Public Key Encryption Schemes

Syntax

A *public-key encryption scheme* PKE consists of the following algorithms:

- A probabilistic, polynomial-time *key generation algorithm* $\mathsf{PKE.KeyGen}$ that on input 1^λ for $\lambda \in \mathbb{Z}_{\geq 0}$, outputs a pair public key/secret key $(\mathsf{PK}, \mathsf{SK})$. The structure of PK and SK depends on the particular scheme.

- A probabilistic, polynomial-time *encryption scheme* $\mathsf{PKE.Encrypt}$ that takes as input 1^λ, a public key PK and a message m, outputs a ciphertext ψ. A ciphertext is a bit string and its structure may depend on the particular scheme;

- A deterministic, polynomial-time *decryption algorithm* $\mathsf{PKE.Decrypt}$ that takes as input 1^λ, a secret key SK and a ciphertext ψ, outputs either a message m or the special symbol reject.

Correctness

A public-key encryption scheme is correct if for all $(\mathsf{PK}, \mathsf{SK}) \in [\mathsf{PKE.KeyGen}(1^\lambda)]$, all m and all $\psi \in [\mathsf{PKE.Encrypt}(\mathsf{PK}, m)]$, we have $\mathsf{PKE.Decrypt}(\mathsf{SK}, \psi) = m$.

Security model

The following attack game is used to define security against adaptive chosen ciphertext attacks.

- The adversary queries a *key generation oracle.* The key generation oracle computes $(\mathsf{PK}, \mathsf{SK}) \overset{R}{\leftarrow} \mathsf{PKE.KeyGen}(1^\lambda)$ and responds with PK.

- The adversary makes a sequence of calls to a *decryption oracle.*

  For each decryption oracle query, the adversary submits a bit string ψ, and the decryption oracle responds with $\mathsf{PKE.Decrypt}(1^\lambda, \mathsf{SK}, \psi)$.

- The adversary submits two messages $m_0, m_1 \in \mathsf{PKE.MSpace}_{\lambda,\mathsf{PK}}$ to the *challenger.* On input m_0, m_1, the challenger then computes $\sigma \overset{R}{\leftarrow} \{0, 1\}$; $\psi^* \overset{R}{\leftarrow} \mathsf{PK.Encrypt}(1^\lambda, \mathsf{PK}, m_\sigma)$; and responds with the target ciphertext ψ^*. In the case of an unrestricted message space, it is required that $|m_0| = |m_1|$.

- The adversary continues to make calls to the *decryption oracle*, subject only to the restriction that a submitted bit string ψ is not identical to ψ^*.

- The adversary outputs $\hat{\sigma} \in \{0, 1\}$, and wins the game if $\hat{\sigma} = \sigma$.

The CCA *advantage of* $\mathcal{A}$ *against* PKE *at* λ, denoted by $\mathsf{AdvCCA}_{\mathsf{PKE},\mathcal{A}}(\lambda)$, to be $|Pr[\sigma = \hat{\sigma} - 1/2]|$ in the above attack game.

Definition 2. *([5])* PKE *is* secure against adaptive chosen ciphertext attack *if* for any probabilistic, polynomial-time adversary $\mathcal{A}$, the function $\mathsf{AdvCCA}_{\mathsf{PKE},\mathcal{A}}(\lambda)$ grows negligibly in λ.

2.4 Decisional Diffie-Hellman Assumption

A computational group scheme $\mathcal{G}$ specifies a sequence S_λ of group distributions. For every value of a security parameter $\lambda \in \mathbb{Z}_{\geq 0}$, S_λ is a probability distribution of group descriptions. A group description Γ specifies a finite abelian group $\hat{G}$, along with a prime-order subgroup G, a generator g of G, and the order q of G. A multiplicative notation for the group operation in $\hat{G}$ is used and the identity element of $\hat{G}$ denoted by 1_G. The notation $\Gamma[\hat{G}, G, g, q]$ indicates that Γ specifies $\hat{G}, G, q$ and q as above [5].

The Decisional Diffie-Hellman is an assumption formulated with respect to a suitable group $\mathcal{G}$ of a large prime order q generated by a given element g. The Decisional Diffie-Hellman assumptions says that is hard to distinguish triples of the form (g^x, g^y, g^z) for random $x, y, z \in \mathbb{Z}_q$ from triples of the form (g^x, g^y, g^{xy}) for random $x, y \in \mathbb{Z}_q$. Formally, let $\mathcal{G}$ be a computational group scheme, specifying a sequence S_λ of group distributions. For all λ, and for all $\Gamma[\hat{G}, G, g, q] \in [S_\lambda]$, the sets $\mathcal{D}_{\lambda,\Gamma}$ and $\mathcal{T}_{\lambda,\Gamma}$ are defined as follows.

$$\mathcal{D}_{\lambda,\Gamma} := \{(g^x, g^y, g^{xy}) \in G^3 : x, y \in \mathbb{Z}_q\}; \tag{2.1}$$

$$\mathcal{T}_{\lambda,\Gamma} := G^3. \tag{2.2}$$

The set $\mathcal{D}_{\lambda,\Gamma}$ is the set of Diffie-Hellman triples. Also, for $\rho \in G^3$, define $\mathsf{DHP}_{\lambda,\Gamma}(\rho) = 1$ if $\rho \in \mathcal{D}_{\lambda,\Gamma}$, and otherwise, define $\mathsf{DHP}_{\lambda,\Gamma}(\rho) = 0$. For all 0/1-valued, probabilistic, polynomial-time algorithms $\mathcal{A}$, and for all λ and all $\Gamma[\hat{G}, G, g, q] \in [S_\lambda]$, the *DDH advantage of* $\mathcal{A}$ *against* $\mathcal{G}$ *at* λ *given* Γ is defined as follows.

Definition 3. *([5]) The decisional Diffie-Hellman assumption for* $\mathcal{G}$ *is that for every probabilistic, polynomial-time 0/1-valued adversary* $\mathcal{A}$*, the function* $\mathsf{AdvDDH}_{\mathcal{G},\mathcal{A}}(\lambda|\Gamma)$ *is negligible in* λ*.*

$$\mathsf{AdvDDH}_{\mathcal{G},\mathcal{A}}(\lambda|\Gamma) := \Big| Pr[\tau = 1 : \rho \leftarrow \mathcal{D}_{\lambda,\Gamma}; \tau \leftarrow \mathcal{A}(1^{\lambda}, \Gamma, \rho)] \\ - Pr[\tau = 1 : \rho \leftarrow \mathcal{T}_{\lambda,\Gamma}; \tau \leftarrow \mathcal{A}(1^{\lambda}, \Gamma, \rho)] \Big|. \tag{2.3}$$

2.5 Cryptographic Hash Functions

A *hashing scheme* HF specifies for each value of λ, a function $\mathsf{HF}^{\lambda} : \mathsf{KeySpace}_{\lambda} \to \mathcal{D}_{\lambda} \to \mathcal{C}_{\lambda}$, which we denote as $\mathsf{HF}^{\lambda}_{hk}(.)$, $hk \in \mathsf{KeySpace}_{\lambda}$, where $\mathsf{KeySpace}_{\lambda}$ is a family of *key spaces* indexed by λ where each key space is a probability space on bit strings, and typically $\mathcal{D}_{\lambda} \subseteq \{0,1\}^*$ and $\mathcal{C}_{\lambda} \subseteq \{0,1\}^{\lambda}$. There must exist a deterministic, polynomial-time algorithm that on input $1^{\lambda}, \mathcal{D}, \mathsf{hk} \in [\mathsf{HF.KeySpace}_{\lambda}]$ and $\rho \in \mathcal{D}$, outputs $\mathsf{HF}^{\lambda}_{\mathsf{hk}}(\rho)$.

Target Collision Resistant hash functions

A family of keyed hash functions is used, such that given a randomly chosen preimage and randomly chosen hash function key, it is computationally infeasible to a probabilistic, polynomial-time adversary to find a different preimage that hashes to the same value using the given hash key.

Definition 4. *([5]) For any probabilistic, polynomial-time algorithm $\mathcal{A}$, if a function $\mathsf{HF}^{\lambda}_{hk}$ is* target collision resistant (TCR) *function then $\mathsf{AdvTCR}_{\mathsf{HF},\mathcal{A}}(\lambda|\mathcal{D})$ is negligible in λ.*

$$\mathsf{AdvTCR}_{\mathsf{HF},\mathcal{A}}(\lambda|\mathcal{D}) := \Pr[\rho \in \mathcal{D} \wedge \rho \neq \rho^* \wedge \mathsf{HF}^{\lambda}_{hk}(\rho^*) = \mathsf{HF}^{\lambda}_{hk}(\rho) : \\ \rho^* \overset{R}{\leftarrow} \mathcal{D}; hk \overset{R}{\leftarrow} \mathsf{HF.Keyspace}_{\lambda}; \\ \rho \overset{R}{\leftarrow} \mathcal{A}(1^{\lambda}, \rho^*, hk)]. \tag{2.4}$$

Universal one-way hash functions

We refer to a Universal One-Way Hash Function (UOWHF) as a slightly stronger security notion compared to target collision resistance as presented above. In UOWHF security the first input to the hash function is chosen adversarially (contrarily to the random sampling presented on the TCR hash function), but independent of the key of the hash function. In this case, the adversary $\mathcal{A}$ should provide another preimage such that when applying a keyed hash function produce the same image.

This definition was formally analyzed in [2], albeit in a concrete security formulation. Let the adversary be defined as $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which consists of two algorithms, $\mathcal{A}_1$ and $\mathcal{A}_2$. First $\mathcal{A}_1$ is run, to produce the first preimage ρ and possibly some extra *state information st*, that $\mathcal{A}_1$ wants to pass to $\mathcal{A}_2$. After $\mathcal{A}_1$ execution, a random key hk is chosen and $\mathcal{A}_2$ is run. $\mathcal{A}_2$ receives as input hk, ρ, st, then it must find a preimage ρ^* different from ρ such that $H_{hk}(\rho) = H_{hk}(\rho^*)$. The value ρ^* can depend on hk but ρ can not.

Definition 5. *([2]) For any probabilistic, polynomial-time algorithm $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$, if $\mathsf{HF}^{\lambda}_{hk}$ is a* universal one-way hash function *then $\mathsf{AdvUOW}_{\mathsf{HF},\mathcal{A}}(\lambda, \mathcal{D})$ is negligible in λ. For any λ and $\mathcal{D}$, $\mathsf{AdvUOW}_{\mathsf{HF},\mathcal{A}}(\lambda, \mathcal{D})$ is defined as*

$$\begin{aligned}\mathsf{AdvUOW}_{\mathsf{HF},\mathcal{A}}(\lambda|\mathcal{D}) := \Pr[\rho \in \mathcal{D} \wedge \rho \neq \rho^* \wedge \mathsf{HF}^{\lambda}_{hk}(\rho^*) = \mathsf{HF}^{\lambda}_{hk}(\rho) :& \\ (\rho^*, st) \leftarrow \mathcal{A}_1(1^{\lambda}), hk \overset{R}{\leftarrow} \mathsf{HF}.\mathsf{Keyspace}_{\lambda};& \qquad (2.5)\\ \rho \overset{R}{\leftarrow} \mathcal{A}_2(hk, st)].&\end{aligned}$$

Chapter 3

Information-theoretically secure authenticated key agreement

In this chapter we analyze an information-theoretically secure authentication key agreement protocol proposed by Dodis et al. in [7]. The authors considered a scenario where two parties *Alice* and *Bob* want to securely communicate over a public channel. The two share a weak-secret, about which an adversary *Eve* has some side information. This adversary Eve is computationally unbounded and has control over the communication channel. The shared weak-secret is modeled as a random variable W arbitrarily distributed over n-size length bitstrings, where Eve has some side-information about it, modeled as a random variable Z correlated to W. Weak-secret W might not be uniformly random, but the adversary must have at least k bits of uncertainty about it. This was formalized using the notion of conditional min-entropy.

We know that information-theoretic secure cryptosystems are often proposed as a theoretic feasibility result, or if practical, substantially less efficient compared to computational cryptosystems with the equivalent security levels. The presented protocol uses as privacy amplification technique a randomness extractor. Concrete constructions of these extractors have been shown to exist ([8]) but they are not practical except for very small parameter sizes. Nevertheless, to get a rough idea of the practicality of this protocol independently of this component, we will simply assume that it can be instantiated using a cryptographic hash function. We will perform a top-down analysis of the components in this protocol to estimate parameter sizes for the protocol's building block structures. With this knowledge we will be able to provide a valid construction of the IT-AKA protocol where the two parties agree on an 128 bit secret key, with 128 bit security.

Preliminaries

The notion of statistical distance between two random variables A, B is defined by $\mathbf{SD}(A, B) = \frac{1}{2}\sum_v |Pr[A = v] - Pr[B = v]|$. We use $A \approx_\varepsilon B$, denoting that A and B are ε-close, as shorthand for $\mathbf{SD}(A, B) \leq \varepsilon$.

Minimal assumptions about the secrecy of W are required [7]. The only requirement is that W has at least k bits of entropy (conditioned on the side-information Z), where k is roughly proportional to the security parameter. This is captured by the notion of an (n, k)-source, which measures the predictability of W. This consists in the measurement of the min-entropy of a weak secret W when sampled according to some joint distribution $(W|Z)$ where the adversary knows Z. In turn, min-entropy measures the predictability of a weak secret W by an adversary.

Definition 6. *([7]) The min-entropy of a random variable W is defined by*

$$H_\infty \overset{def}{=} -log(max_w Pr[W = w])\,.$$

Conditioned predictability must be considered when W is sampled according to some joint distribution $(W|Z)$ where the adversary sees Z.

Definition 7. *([7]) The average conditional min-entropy is defined by*

$$\tilde{H}_\infty(W|Z) \overset{def}{=} -log(\mathbb{E}_{z \leftarrow Z} max_w Pr[w = W|z = Z]).$$

Definition 8. *([7]) For some joint distribution $(W|Z)$ is a (n, k)-source if W takes values over $\{0, 1\}^n$ and its min-entropy is at least k iff $Pr[X = x] \leq 2^{-k}$ for all x.*

3.1 Definitions and Security Model

The goal of an authenticated key agreement protocol is the following. Alice and Bob share a secret W about which Eve has some side-information Z. They would like to run a protocol, in which they agree on a *shared random key.* Alice and Bob each have two candidate keys r_A, r_B that at the beginning are set to the special value $\perp$. Only after Alice reaches `KeyDerived` state, can Bob reach `KeyConfirmed` state. When these states are reached, the respective candidate key its set as some l-bit value (not $\perp$) and it is not modified after that event. This key is to be used in some cryptographic task, so the parties would like to be assured that the key is and will remain private. At the same time this protocol allows that sharing the key and its preparation or the sending of the authenticated-ciphertext are not necessarily synchronous actions. The advantage to this asymmetric definition is that

Alice is able to use the key r_A for some cryptographic task at the point she reaches `KeyDerived`. Then, when Bob receives some message from Alice, he can reach `KeyConfirmed` alone. This definition generalizes the definitions for one-round key agreement protocols where Alice obtains a key on her own, goes into the `KeyDerived` state and sends a single message to Bob [7].

The notions of `KeyDerived` and `KeyConfirmed` are described informally as follows. If Alice reaches `KeyDerived` state it means that she has a uniformly *random candidate key* r_A, which is kept private no matter the behavior of the adversary during the protocol. However, she is not sure if her key is shared with Bob, or if he is even involved in the protocol execution. If Bob reaches `KeyConfirmed` state and obtains a candidate key r_B, this means that Alice must have been involved in that particular execution of the protocol and she reached `KeyDerived` state. In this point, both have a shared key where $r_A = r_B$ which is private from any adversary [7]. Formally, this intuition is presented in the following definition.

Definition 9. *([7]) In a $(n, k, l, \varepsilon, \delta)$-(information theoretic) authenticated key agreement protocol (*$\mathsf{IT\text{-}AKA}$*), Alice and Bob have candidate keys $r_A, r_B \in \{0,1\}^l \cup \{\perp\}$ respectively. For any active adversarial strategy $\mathcal{A}$ employed by Eve, let R, R' be random variables which denote the values of the candidate keys r_A, r_B at the conclusion of the protocol execution and let T be a random variable which denotes the transcript of the entire protocol as seen by Eve. The protocol must have the following three properties:*

***Key privacy**: If $(W|Z)$ is (n, k)-source then, for any adversarial strategy $\mathcal{A}$ employed Eve, if Alice reaches the* `KeyDerived` *state during the protocol execution, the $(Z, T, R) \approx_\varepsilon (Z, T, U_l)$.*

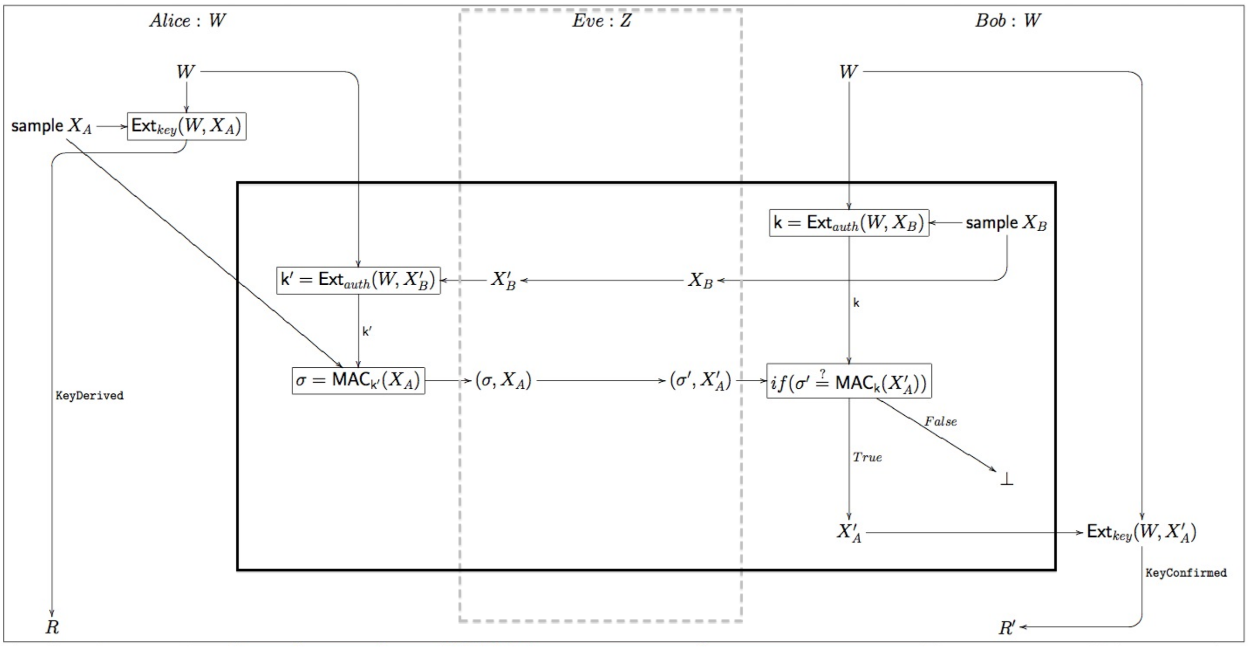
***Key Authenticity**: We say that the protocol has pre-application authenticity if for any (n, k)-source $(W|Z)$ and any adversarial strategy $\mathcal{A}$ employed by Eve, the probability that Bob reaches the* `KeyConfirmed` *state and $R \neq R'$ is at most δ. We say that the protocol has post-application authenticity if the above holds even if the adversary is given R immediately after Alice reaches* `KeyDerived` *state [7].*

***Correctness**: If Eve is passive, then Alice reaches the* `KeyDerived` *state, Bob reaches the* `KeyConfirmed` *state, and $R = R'$ (with probability 1).*

The notion of pre/post-application authenticity was generalized from a previous work by Dodis, Katz, Reyzin and Smith [6], where it was noted that, if Alice wants to use her key r_A immediately after reaching KeyDerived, she needs the assurance that her use of the key does not help the adversary Eve break authenticity.

3.2 A concrete IT-AKA protocol

As we want to create an actual implementation, we focus on a particular IT-AKA protocol presented in [7]. In this protocol, Alice reaches KeyDerived state by simply using a randomness extractor Ext_{key} where a random seed X_A and the weak secret W are used to derive R. The main idea behind this construction consists in Alice using a message authentication protocol to transfer this random seed X_A to Bob, so that both parties can use a randomness extractor to obtain a shared key from the weak randomness source. The difficulty of the problem lies in using the same weak randomness source to feed the message authentication protocol that protects X_A, and also to derive the final shared key: the adversary Eve can potentially learn some information about W during the course of the authentication protocol, thus compromising the secrecy of the final key. In this construction, this problem is solved by employing an interactive message authentication protocol with *look-ahead* security. Overall the protocol displays the structure shown in Figure 3.1.



$Alice : W$
$Eve : Z$
$Bob : W$
W
W
sample X_A
$\mathsf{Ext}_{key}(W, X_A)$
$\mathsf{k} = \mathsf{Ext}_{auth}(W, X_B)$
sample X_B
$\mathsf{k}' = \mathsf{Ext}_{auth}(W, X'_B)$
X'_B
X_B
k
k'
$\sigma = \mathsf{MAC}_{\mathsf{k}'}(X_A)$
(σ, X_A)
(σ', X'_A)
$if(\sigma' \overset{?}{=} \mathsf{MAC}_{\mathsf{k}}(X'_A))$
KeyDerived
False
True
$\perp$
X'_A
$\mathsf{Ext}_{key}(W, X'_A)$
KeyConfirmed
R
R'

Figure 3.1: Structure of the IT-AKA protocol presented in [7]. The internal bold box depicts a message authentication protocol. The dotted box shows what values can the active adversary Eve access and modify.

An intuitive explanation of the security is described as follows. The message authentication protocol guarantees that $X'_A = X_A$ if Bob reaches KeyConfirmed, implying that $R = R'$ and yielding authenticity, even if the adversary Eve has access to R. For privacy, the only information that an active adversary might obtain about the weak secret W is the tag $\sigma = \mathsf{MAC}_{\mathsf{k}'}(X_A)$. Due to the characteristics of the MAC, σ is fully determined by k' and X_A, and therefore is independent of W when conditioned on k'. This means that keys R, R' are secure as long there is enough entropy left in W conditioned on k' and Z [7]. This is ensured by the message authentication protocol. In the following, we will describe in detail the individual components used in the protocol.

3.3 Randomness Extractor

Randomness extractors are functions that convert weak random sources, which may have biases and correlations, into almost-perfect random sources. If only a weak random source W is used this task would be impossible, so the extractor is provided with a short seed X of truly random bits to help with the extraction [8].

Definition 10. *([8]) A function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^l$ is a $(n, k, d, l, \varepsilon)$-extractor if for any $W \in \{0,1\}^n$ with at least k bits of min-entropy, and a random source $X \in \{0,1\}^d$ we can extract a $R = \mathsf{Ext}(W; X)$, where $R \in \{0,1\}^l$ is ε-close to an uniformly random value.*

3.4 Message Authentication Protocol

A special Message Authentication Protocol was proposed in [7] to authenticate the short seed used in the key agreement protocol. In this protocol, Alice and Bob share a weak secret W. Alice sends an authenticated message μ_A to Bob, in the presence of Eve who has complete control over the network and can modify protocol messages arbitrarily. Bob should either receive μ_A or detect an active attack and quit by outputting $\perp$.

Definition 11. *([7]) An (n, k, m, δ)-message authentication protocol* AUTH *is a protocol in which Alice starts with a source message $\mu_A \in \{0,1\}^m$ and, at the conclusion of the protocol, Bob outputs a received message $\mu_B \in \{0,1\}^m \cup \{\perp\}$. The following properties are required.*
Correctness. *If the adversary Eve is passive then, for any source message $\mu_A \in \{0,1\}^m$, $Pr[\mu_B = \mu_A] = 1$.*
Security. *If $(W|Z)$ is an (n, k)-source then, for any source message $\mu_A \in \{0,1\}^m$ and any active adversarial strategy employed by Eve, $Pr[\mu_B \neq \{\mu_A, \perp\}] \leq \delta$.*

If W was a perfectly random secret, we could use a standard message authenticated code (MAC) and only a single round where Alice would send her message μ_A along with a tag $\sigma = \mathsf{MAC}_W(\mu_A)$. This strategy does not extend in general to the case of weak secrets, for which theoretical lower bounds on communication complexity imply that one-round protocols are either impractical or impossible [7]. The proposed protocol therefore follows a challenge-response structure: it begins with Bob sending a random challenge X to Alice, who then uses secret W to compute a response that will authenticate her message. The random challenge X is used as seed for some extractor Ext. If the adversary does not modify the seed, then Alice will be able to derive a shared random key $R = \mathsf{Ext}(W; X)$. Alice can then authenticate her message μ_A, by using R as a key for a message authentication code MAC and sending the tag $\sigma = \mathsf{MAC}_R(\mu_A)$ along with μ_A as her response to Bob. However, this construction is not secure in general: Eve can modify the extractor seed X to some arbitrarily related X', causing Alice to derive some incorrect, but possible related key $R' = \mathsf{Ext}(W, X')$. The proposed approach therefore relies on special components to instantiate the challenge-response protocol: a *look-ahead* extractor plugged in to an *look-ahead* MAC.

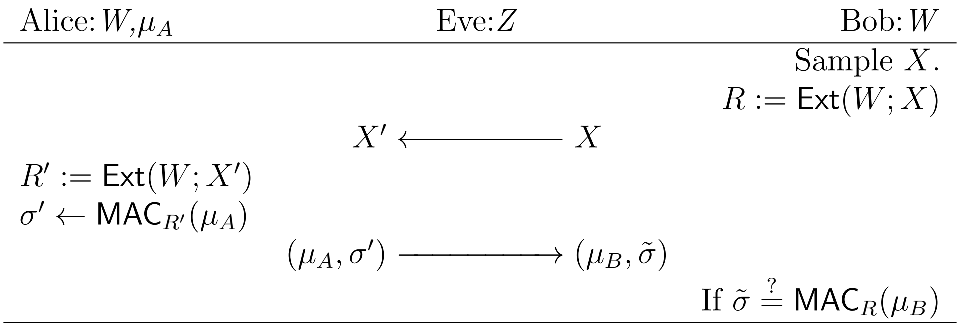

Alice: W,μ_A
Eve:Z
Bob: W
Sample X.
$R := \mathsf{Ext}(W; X)$
$X' \longleftarrow X$
$R' := \mathsf{Ext}(W; X')$
$\sigma' \leftarrow \mathsf{MAC}_{R'}(\mu_A)$
$(\mu_A, \sigma') \longrightarrow (\mu_B, \tilde{\sigma})$
If $\tilde{\sigma} \overset{?}{=} \mathsf{MAC}_R(\mu_B)$

Figure 3.2: Framework for Message Authentication Protocols. [7]

Theorem 1. *([7]) Plugging in a look-ahead extractor with a* MAC *with look-ahead security an efficient two-round (n, k, m, δ)-message authentication protocol is constructed for any integers $n \geq k, m$ and any $\delta \geq 0$ as long as $k > O(m(m + log(n) + log(1/\delta)))$. Subsequently, the* MAC *key is bounded by $\tau = 4m(m + log(1/\delta))$.*

By getting an upper-bound of τ we have the required guarantees about the entropy loss on W. The parameters reached for this protocol were considered vastly sub-optimal for all but very short messages [7], yet this is all that is required in the key agreement protocol.

3.4.1 Look-ahead extractors

A look-ahead extractor is a function that uses a random seed X to extract an arbitrary number of blocks of randomness $R_1, \ldots, R_t$ from some secret W. If we take some seed X' that is arbitrarily related to X, and use it to extract blocks $R'_1, \ldots, R'_i$ from W, then any suffix $R_{i+1}, \ldots, R_t$ of the sequence extracted with X will look uniformly random, even when given the prefix $R'_1, \ldots, R'_i$ in the related sequence. In this kind of extraction the adversarial entity Eve is not capable of modifying the seed in such way that the incorrectly extracted blocks could permit "looking-ahead" into the original sequence of blocks [7].

Definition 12. *([7]) Let* $la\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to (\{0,1\}^l)^t$ *be a function such that* $la\mathsf{Ext}(W;X)$ *outputs blocks* $R_1, \ldots, R_t$ *with* $R_i \in \{0,1\}^l$*.* $la\mathsf{Ext}$ *is a* (n,k,d,l,t,ε)*-look-ahead extractor if, for any* (n,k)*-source* $(W|Z)$*, any adversarial function* $\mathcal{A}$ *and any* $i \in \{0, \ldots, t-1\}$*,*

$$(Z, X, [R'_1, \ldots, R'_i], [R_{i+1}, \ldots, R_t])) \approx_\varepsilon (Z, X, [R'_1, \ldots, R'_i], U_{l(t-i)}) \tag{3.1}$$

where $[R_1, \ldots, R_t] = la\mathsf{Ext}(W;X)$*,* $X' = \mathcal{A}(X,Z)$*,* $[R'_1, \ldots, R'_t] = la\mathsf{Ext}(W;X')$ *and* U_l *denotes a uniformly random distribution over* l *bit strings.*

The concrete construction of a look-ahead extractor is as follows and uses an *alternating extraction* procedure [7]. This is an iterative process which uses two randomness extractors, Ext_q and Ext_w, and runs in t iterations. The look-ahead extractor's seed X is interpreted as $X = (Q, S_1)$, where Q is used as random seed for randomness extractor Ext_q and S_1 is used as random seed for Ext_w. In the first iteration, the initial seed S_1 is used by Ext_w, $R_1 := \mathsf{Ext}_w(W;S_1)$ is computed and R_1 is used by Ext_q; where Ext_q then computes $S_2 := \mathsf{Ext}_q(Q;R_1)$. In each subsequent iteration i, Ext_q outputs S_i which is used by Ext_w; this outputs $R_i := \mathsf{Ext}_w(W;S_i)$, and then Ext_q goes on to computes $S_{i+1} := \mathsf{Ext}_q(Q;R_i)$. The two extractors together produce the following sequence.

$$S_1, R_1 = \mathsf{Ext}_w(W;S_1), S_2 = \mathsf{Ext}_q(Q;R_1), \ldots, S_t = \mathsf{Ext}_q(Q,R_{t-1}), R_t = \mathsf{Ext}_w(W,S_t) \tag{3.2}$$

In the final iteration, this protocol outputs all blocks calculated by Ext_w, i.e. the sequence $R_1, \ldots, R_t$ is also the look-ahead extractor's output. A relation between the characteristics from the randomness extractors used and the resulting look-ahead extractor is described in the following theorem.

Theorem 2. *([7]) Given an* $(n_w, k_w - (2l)t, l, l, \varepsilon_w)$*-extractor* Ext_w *and a* $(n_q, n_q - (2l)t, l, l, \varepsilon_q)$*-extractor* Ext_q*, the construction yields an* $(n_w, k_w, n_q + l, l, t, t^2(\varepsilon_w + \varepsilon_q))$*-look-ahead extractor.*

3.4.2 Look-ahead MAC

The look-ahead MAC is a special kind of message authentication code. Eve has the ability of modifying the look-ahead extractor seed during the initial flow, so then she can perform some limited related key attack. The look-ahead MAC is a message authentication code that is secure under the types of related key attacks allowed by the look-ahead extractor.

Definition 13. *([7]) A family of functions* $\{\mathsf{MAC}_r : \{0,1\}^m \to \{0,1\}^s\}$ *indexed by the keys* $r \in (\{0,1\}^l)^t$ *is a* $(m, s, l, t, \varepsilon, \delta)$*-MAC with look-ahead security if, for any random variables* $R = [R_1, \ldots, R_t], R' = [R'_1, \ldots, R'_t]$, $V = (X, Z)$, *which satisfy the look-ahead property:*

$$(V, [R'_1, \ldots, R'_i], [R_{i+1}, \ldots, R_t]) \approx_\varepsilon (V, [R'_1, \ldots, R'_i], U_{(t-i)l}) \quad \forall i \in \{0, \ldots, t-1\} \tag{3.3}$$

and any $\mu_A \in \{0,1\}^m$ *and any adversarial function* $\mathcal{A}$*, we have*

$$Pr\left[\mu_B \neq \mu_A, \mathsf{MAC}_R(\mu_B) = \tilde{\sigma} \middle| \begin{array}{l} \sigma' \leftarrow \mathsf{MAC}_{R'}(\mu_A) \\ (\mu_B, \tilde{\sigma}) \leftarrow \mathcal{A}(V, \sigma') \end{array}\right] \leq \delta. \tag{3.4}$$

Construction

Any collection of pairwise top-heavy sets can be used to construct a MAC with look-ahead security.

Definition 14. *([7]) Given* $S_1, S_2 \subseteq \{1, \ldots, t\}$*, we say that the ordered pair* (S_1, S_2) *is top-heavy if there is some integer* j *that,* $|S_1^{\geq j}| > |S_2^{\geq j}|$*, where* $S^{\geq j} \overset{def}{=} \{s \in S | s \geq j\}$*. It is possible that* (S_1, S_2) *and* (S_2, S_1) *are both top-heavy. For a collection* Ψ *of sets* $S_i \subseteq \{1, \ldots, t\}$ *we say that* Ψ *is pairwise top-heavy if every ordered pair* (S_i, S_j) *of sets* $S_i, S_j \in \Psi$ *with* $i \neq j$*, is top-heavy.*

Lemma 1. *([7]) Assume that a collection* $\Psi = \{S_1, \ldots, S_{2^m}\}$ *of sets* $S_i \subseteq \{1, \ldots, t\}$ *is pairwise top-heavy. Then the family of functions* $MAC_r(\mu) \overset{def}{=} [r_i | i \in S_\mu]$*, indexed by* $r \in (\{0,1\}^l)^t$*, is a* $(m, s, l, t, \varepsilon, \delta)$*-MAC with look-ahead security where* $s = l \ \max_{S_i \in \Psi}(|S_i|)$*,* $\delta \leq (2^{m-l} + 2^m \varepsilon)$*. Furthermore, if there is an efficient mapping of* $\mu \in \{0,1\}^m$ *to* S_μ*, then the construction is efficient.*

To construct efficient MACs with look-ahead security, a large collection of sets which is pairwise top-heavy must be constructed. A collection Ψ is obtained by mapping a m bit message $\mu = (b_1, \ldots, b_m) \in \{0,1\}^m$ to a subset $S \subseteq \{1, \ldots, 4m\}$ using the function

$$f(b_1, \ldots, b_m) \overset{def}{=} \{4i - 3 + b_i, 4i - b_i | i = 1, \ldots, m\} \tag{3.5}$$

i.e. each bit b_i decides if to include the values $\{4i-3, 4i\}$ (if $b_i = 0$) or the values $\{4i-2, 4i-1\}$ (if $b_i = 1$).

Lemma 2. *([7]) The above construction gives us a pairwise top-heavy collection Ψ of 2^m sets $S \subseteq \{1, \ldots, t\}$ where $t = 4m$. Furthermore, the function f is an efficient mapping of $\mu \in \{0,1\}^m$ to S_μ.*

Corollary 1. *([7]) We get an $(m, s, l, t, \varepsilon, \delta)$-$\mathsf{MAC}$ with look-ahead security for any m, l, ε with $t = 4m$, $s = 4ml$ and $\delta \leq (2^{m-l} + 2^m\varepsilon)$.*

3.5 Global concrete security analysis

The building blocks presented in the previous sections, are combined into an information theoretically secure authenticated key agreement protocol under the conditions formalized in the following theorem.

Theorem 3. *([7]) Let AUTH be an (n, k, m, δ)-message authentication protocol as above using components Ext_{auth} and MAC such that key size for MAC is τ bits long. Let Ext_{key} be an $(n, k-\tau, d = m, l, \varepsilon)$-extractor. Then the construction in Figure 3.1 is an $(n, k, l, \varepsilon, \delta)$-$\mathsf{IT\text{-}AKA}$ with pre-application authenticity. Assuming that AUTH is an $(n, k-l, m, \delta)$-message authentication protocol, then we get post-application authenticity.*

Based on this result, we have compiled the information about bit-length requirements, statistical distances and probabilities of impersonation, in order to instantiate and implement the construction for some practically meaningful values. We have hierarchically organized the requirements in a top-down structure, where at the top we set the IT-AKA protocol and at the bottom the randomness extractors used as building blocks. This is depicted in Figure 3.3. At the top of this hierarchy we set some preliminary restrictions, particularly the size of the secret key agreed (denoted by $R, R' \in \{0,1\}^l$ where $l = 128$ bits), statistical distance from a uniform distribution (denoted $\varepsilon = 2^{-128}$), and probability of impersonation in the authentication protocol (denoted $\delta = 2^{-128}$).

Variables n, m and l' bitlengths are free from restrictions, so we modeled them in order to present three case studies. We present an optimal, average, and a worst case situation. The optimal case depicts a good source of randomness where W has 93.2% of truly random bits. The worst case represents a bad source of randomness where W has only 14.2% of truly random bits. The average case depicts an average source of randomness where 56.3% are random bits.

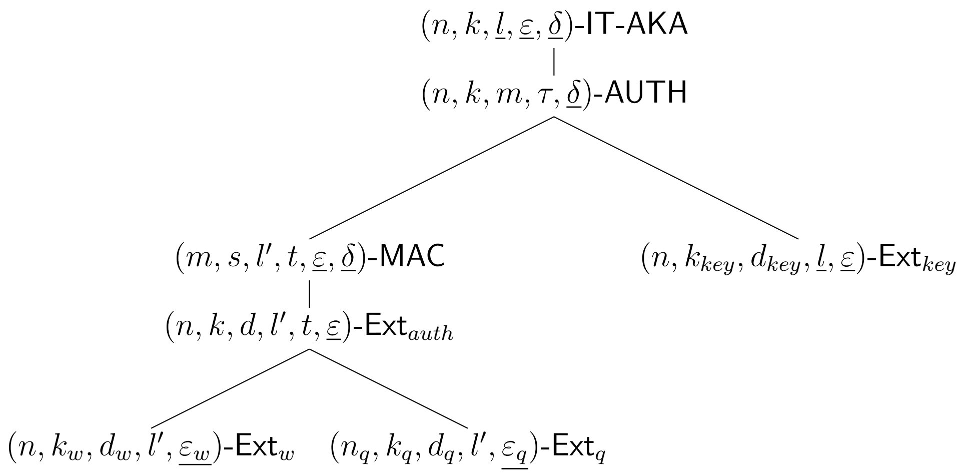

$(n, k, \underline{l}, \underline{\varepsilon}, \underline{\delta})$-IT-AKA
$(n, k, m, \tau, \underline{\delta})$-AUTH
$(m, s, l', t, \underline{\varepsilon}, \underline{\delta})$-MAC
$(n, k_{key}, d_{key}, \underline{l}, \underline{\varepsilon})$-Ext$_{key}$
$(n, k, d, l', t, \underline{\varepsilon})$-Ext$_{auth}$
$(n, k_w, d_w, l', \underline{\varepsilon_w})$-Ext$_w$
$(n_q, k_q, d_q, l', \underline{\varepsilon_q})$-Ext$_q$

Figure 3.3: IT-AKA's building block hierarchy. The underlined variables are fixed a priori.

A special note for weak secret $W \in \{0,1\}^n$, where nothing more is required than $n \geq k$ throughout all the requirements presented. Variable k defines the bits of min-entropy. With only this requirement, our calculations tend to match the minimum length required for W that will be $n = k$. Of course, in a real-world application, fixing k will imply a possibly much larger value for n. The chosen lengths for n, m and l' are presented in table 3.1.

	n	m	l'	ratio k/n (%)	
1	2^{18}	30	32	14.2	worst case
2	2^{15}	15	64	56.3	average
3	2^{15}	25	72	93.2	optimal

Table 3.1: The chosen values for the free variables for the four cases presented.

We stress that the results presented in the following sections are just indicative, as they are based on the assumption that efficient randomness extractors can be constructed for any parameter sizes. However, for the concrete construction suggested in [7], which is the extractor based in [8], obtaining an efficient implementation is not feasible for the large values of n that we are considering in this analysis, as this would imply dealing with irreducible polynomials of order n.

Randomness Extractor Ext$_{key}$

This randomness extractor Ext$_{key}$: $\{0,1\}^n \times \{0,1\}^{d_{key}} \rightarrow \{0,1\}^l$ is responsible for the extraction of $R = R' = \text{Ext}_{key}(W; X_A)$, the secret key to be agreed by the parties. Its output length is set a priori as $l = 128$. The random seed $X_A \in \{0,1\}^{d_{key}}$ where $d_{key} = m$ and where m is a free parameter corresponding to the size of the random seed used in the secret key extraction procedure, which has an impact in the setting of other parameters. The bits of min-entropy k_{key} are the determined by $k_{key} = k - \tau$ where k represents the entropy needed in AUTH message authentication protocol, described in Theorem 3, and the results presented in tables 3.9 and 3.10. The notation *pre* and *post* for k present in table 3.2 describes the lengths required for the two types of authentication guarantees (pre and post-application authenticity) for AUTH.

	n	k (*pre*)	k (*post*)	d	l	$log(1/\varepsilon)$
1	$2^{18} = 262144$	18349	18477	30	128	128
2	$2^{15} = 32768$	9884	10012	15	128	128
3	$2^{15} = 32768$	15227	15355	25	128	128

Table 3.2: Bitlengths for the construction of randomness extractor Ext_{key}.

MAC with look-ahead security

The function $\mathsf{MAC_k} : \{0,1\}^m \rightarrow \{0,1\}^s$ takes a message $X_A \in \{0,1\}^{d_{key}=m}$, where m is the same bitlength used in the seed to Ext_{key}, and outputs a tag $\sigma \in \{0,1\}^s$ where $s = 4ml'$. The key $\mathsf{k} \in (\{0,1\}^{l'})^t$ is the output of a look-ahead extractor, where $t = 4m$, and l' is a bitlength shared by the MAC, the look-ahead extractor Ext_{auth}, and the two randomness extractors Ext_q and Ext_w used as building blocks of Ext_{auth}. As we said before, the probability of impersonation δ is set to 2^{-128}, with the requirement from the MAC definition $\delta \leq (2^{m-l'} + 2^m\varepsilon)$ which is true for all values m, l', ε displayed in table 3.3.

	m	s	l'	t	$log(1/\varepsilon)$	$log(1/\delta)$
1	30	3840	32	120	128	128
2	15	3840	64	60	128	128
3	25	7200	72	100	128	128

Table 3.3: Bitlengths to construct a look-ahead MAC.

Look-ahead Extractor

The look-ahead extractor $\mathsf{Ext}_{auth} : \{0,1\}^n \times \{0,1\}^d \rightarrow (\{0,1\}^{l'})^t$, outputs $\mathsf{k} = \mathsf{Ext}_{auth}(W; X_B)$ taking as arguments the weak secret $W \in \{0,1\}^n$ and the random seed (challenge) $X_B \in \{0,1\}^d$. The min-entropy k for some value W has two different requirements, one given by the look-ahead extractor construction (Theorem 2) as $k = k_w$, where k_w is the min-entropy of the weak secret used in extractor Ext_w; and the other argument given by a feasibility results, which establishes the minimum value for k as

$$k \geq 2(t+2)max(l', \mathcal{O}(log(n)+log(t)+log(1/\varepsilon)))\,.$$

We therefore calculate k as follows.

$$k = max\begin{cases} k_w \\ 2(t+2)max(l', \mathcal{O}(log(n)+log(t)+log(1/\varepsilon))) \end{cases} \tag{3.6}$$

The other Ext_{auth}'s argument is a random seed $X_B \in \{0,1\}^d$, where by definition $d = n_q + l'$, with $d \geq \mathcal{O}(t(l' + log(n) + log(t) + log(1/\varepsilon)))$. We use the same strategy as before to reach the minimum length d that assured the previous condition.

$$d = max\begin{cases} n_q + l' \\ \mathcal{O}(t(l' + log(n) + log(t) + log(1/\varepsilon))) \end{cases} \tag{3.7}$$

Length l' is the same as the one used in the MAC construction. Variable t defines the number of iterations between the two extractors. This value comes from our analysis of the MAC construction, previously described in table 3.3. The results to a practical look-ahead extractor for our requirements are presented in table 3.4.

	n	k	d	l'	t	$log(1/\varepsilon)$
1	$2^{18} = 262144$	37309	22189	32	120	128
2	$2^{15} = 32768$	18464	12774	64	60	128
3	$2^{15} = 32768$	30527	22164	72	100	128

Table 3.4: Bitlengths to construct a look-ahead extractor Ext_{auth}.

We now look at the two randomness extractors Ext_w and Ext_q used as components of the lookahead extractor, as described in Theorem 2. These two extractors have slightly different definitions. The size of the weak secret used by Ext_q is $Q \in \{0,1\}^{n_q}$ where $n_q = d - l'$. The size of weak secret W size is set as described before. The requirements of min-entropy for each weak secret used in each randomness extractor are different. In the case of Ext_q it is given by $k_q = n_q - (2l')t$, whereas in the case of Ext_w it is given by $k_w = k - (2l')t$. Finally, the random seeds needed in both extractors $X \in \{0,1\}^d$ are $d = l'$, where l' is also the output size for these extractors, for the look-ahead extractor Ext_{auth} and for the MAC.

From above, we have the statistical distance of the lookahead extractor set at $\varepsilon = 2^{-128}$. In Theorem 2 the definition of ε directly relates to the statistical distances from the two extractors Ext_q and Ext_w and the number of iterations t as $\varepsilon = t^2(\varepsilon_w + \varepsilon_q)$. We calculate these two statistical distances from the deconstruction of Ext_{auth}'s statistical distance ε. We define $t = 2^X$ and the statistical distances $\varepsilon_q = \varepsilon_w = 2^{-E}$, where E is calculated as follows:

$$\begin{aligned}
\varepsilon = t^2(\varepsilon_w + \varepsilon_q) &= 2^{-128};\\
2^{2X}(2^{-w} + 2^{-q}) &= 2^{-128};\\
2^{2X}(2^{-E} + 2^{-E}) &= 2^{-128};\\
2^{2X+1-E} &= 2^{-128};\\
2X - E + 1 &= -128;\\
E &= 2X + 128 + 1;\\
E &= 2(log(t)) + 128 + 1.
\end{aligned}$$

Our conclusions are presented in tables 3.5 and 3.6.

	n	k_w	d_w	l'	$\varepsilon_w = log(1/E)$
1	$2^{18} = 262144$	29629	32	32	143
2	$2^{15} = 32768$	10784	64	64	141
3	$2^{15} = 32768$	16127	72	72	142

Table 3.5: Bitlengths to construct randomness extractor Ext_w.

	n_q	k_q	d_q	l'	$\varepsilon_q = log(1/E)$
1	22157	14477	32	32	143
2	12710	5030	64	64	141
3	22092	7692	72	72	142

Table 3.6: Bitlengths to construct randomness extractor Ext_q.

Message authentication protocol

In Theorem 1 to construct an efficient (n, k, m, δ)-message authentication protocol it is required that $n \geq k, m$, that the impersonation probability $\delta \geq 0$, and that $k > O(m(m + log(n) + log(1/\delta)))$. The size of the MAC key is upper bounded by $\tau = 4m(m + log(1/\delta))$ [7]. These bit lengths are described in tables 3.7 and 3.8.

	n	k	m	τ	$log(1/\delta)$
1	$2^{18} = 262144$	37309	30	18960	128
2	$2^{15} = 32768$	18464	15	8580	128
3	$2^{15} = 32768$	30527	25	15300	128

Table 3.7: AUTH parameters for pre-application authenticity.

Key agreement protocol

Theorem 3 states that an (n, k, m, δ)-message authentication protocol which uses the components Ext_{auth} and MAC with look-ahead security coupled with an $(n, k -$

	n	k	m	τ	$log(1/\delta)$
1	$2^{18} = 262144$	37437	30	18960	128
2	$2^{15} = 32768$	18592	15	8580	128
3	$2^{15} = 32768$	30655	25	15300	128

Table 3.8: AUTH parameters for post-application authenticity.

$\tau, d_{key}, l, \varepsilon)$-$\mathsf{Ext}_{key}$ randomness extractor is a $(n, k, l, \varepsilon, \delta)$-IT-AKA with pre-application authenticity. To achieve post-application authenticity we need to construct an $(n, k-l, m, \delta)$-message authentication protocol instead. All values for both pre and post-application authenticity are described in the tables 3.9 and 3.10 respectively.

	n	k	l	$log(1/\varepsilon)$	$log(1/\delta)$
1	$2^{18} = 262144$	37309	128	128	128
2	$2^{15} = 32768$	18464	128	128	128
3	$2^{15} = 32768$	30527	128	128	128

Table 3.9: IT-AKA parameter sizes for pre-application authenticity.

	n	k	l	$log(1/\varepsilon)$	$log(1/\delta)$
1	$2^{18} = 262144$	37437	128	128	128
2	$2^{15} = 32768$	18592	128	128	128
3	$2^{16} = 32768$	30655	128	128	128

Table 3.10: IT-AKA parameter sizes for post-application authenticity.

3.6 Practical Implementation

We developed a C++ program that simulates the IT-AKA protocol and produces 2^{18} bit random string as the weak secret. As pointed out before there is no actual randomness extractor construction, so to this purpose we have created an extractor based on the SHA-256 hash function. The extractor, as described produces a 256-bit output from the n-bit weak source.

From this randomness extractor and based on the construction proposed before, we have created a function that mimics the behavior of a look-ahead extractor. All other functions not directly related with the randomness extractor, were constructed according to the construction proposed.

Even with optimized code using the optimization flag `gcc -O2` each execution is memory and CPU intense. Our performance tests were made on a machine with a two core 2.4 Ghz *Intel Core 2 duo* processor with 5GB of RAM. The average CPU loads were near 100% and consumed approximately 2.7GBytes of memory. The execution times are presented in 3.11.

time	1m43.364s
	1m44.900s
	1m53.591s

Table 3.11: Execution times of IT-AKA's C++ implementation.

Chapter 4

Computationally secure password authenticated key exchange

Protocols for password-based authentication key exchange (PAKE) enable two parties who share a short, low-entropy password to agree on a cryptographically strong session key. Password-based protocols for authenticated key exchange are designed to work even if the passwords used are drawn from a small space in which an adversary might enumerate all the possible passwords off-line.

In this scenario there are two entities – a client C and a server S – where C holds a password pw and S holds a key related to this. These parties engage in a conversation at the end of which each holds a session key sk, which is private and only known by the two of them. A third entity is present which is an active adversary $\mathcal{A}$ whose capabilities include enumerating, off-line, the words in some dictionary which is rather likely to include pw. In a protocol considered good, the adversary's chance to defeat the protocol's goals will depend on how much $\mathcal{A}$ interacts with protocol participants, and will not significantly depend on its off-line computing time.

This protocol problem was originally suggested by Bellovin and Merritt [3], who also offered a protocol named Encrypted Key Exchange (EKE), and an informal security analysis.

4.1 Definitions and Security Model

Consider a fixed set of participants (denoted as Principals) each of which is either a client $C \in \mathit{Client}$ or a server $S \in \mathit{Server}$ (Client and Server are disjoint). Let User be a set with all the elements of Client and Server. Each client $C \in \mathit{Client}$ has a password pw_C. Each $S \in \mathit{Server}$ holds a vector $PW_S = \langle pw_C \rangle_{C \in \mathit{Client}}$ which contains an entry for each client. The password pw_C is chosen from a relatively small space of possible passwords and is used by the client C to authenticate the connection.

In [1], passwords pw_C, pw_S are described as long-lived keys. The long-lived keys are generated by a long-lived key generator PW. A possibility for PW is to determinate a password for a client $A \in Client$ as $pw_A \overset{R}{\leftarrow} PW_A$, for some finite set PW_A and $pw_B[A]_{B \in Server}$ is set as pw_A.

Initialization

An Initialization phase occurs where all the public parameters and passwords needed for the instances of the protocol are created. During an execution a Principal can run many instances, each one of them only once. An instance is denoted as Π^i_U where $i \in N$ and $U \in User$. In this phase, for all i for every principal U the parameters sid^i_U, pid^i_U, the session key sk^i_U and the state preserving values $state$, $accu^i_U$, $term^i_U$, $used^i_U$ are initiated. Session id (sid^i_U) is used to uniquely name an occurring session for some Principal U. Partner id (pid^i_U) names the Principal with which the instance believes it has just exchanged a key. sid^i_U and pid^i_U are write-only and public. Session key sk^i_U is also write-only but secret, and for each client and server instances at most one value is accepted. The state preserving values $state$, $accu^i_U$, $term^i_U$, $used^i_U$ are initialized. These values are used to maintain a global state throughout the execution for an instance i. The passwords for each client $C \in Client$ are generated independently, uniformly and randomly from a relatively small finite set, as described above [1].

Execution of the Protocol

Principals behave in response to input from their environment. In the formal model, these inputs are provided by the adversary [11]. Each principal is able to execute the protocol multiple times with different partners. This is modeled by allowing each principal an unlimited number of instances in which to execute the protocol. Any instance is used only once. We assume that the adversary has complete control over all communications, by that the adversary's interaction with the principals is modeled via access to oracles whose inputs may range over $U \in User$ and $i \in N$.

The adversary's interaction with the principals is modeled via access to the following oracles:

- $Send(U, i, M)$ - This sends message M to instance Π^i_U. The oracle runs this instance as in a real execution, maintaining state as appropriate. The output of Π^i_U is given to the adversary;

- $Execute(C, i, S, j)$ - This oracle executes the protocol between instances of Π^i_U and Π^i_S, where $C \in$ *Client* and $S \in$ *Server*, and outputs a transcript of this execution. This transcript includes everything an adversary would see when eavesdropping on a real-world execution of the protocol;

- $Reveal(U, i)$ - This outputs the session key sk^i_U (stored as part of the global state) of instance Π^i_U;

- $Test(U, i)$ - This query allowed only once, at any time during the adversary's execution. A random bit b is generated; if $b = 1$ the adversary is given sk^i_U, and if $b = 0$ the adversary is given a random session key;

- $Corrupt(U, pw)$ - The adversary obtains pw_U and the states of all instances of U. This is a very damaging type of query. This query models the possibility of subverting a principal by witnessing a user type in his password, installing a *trojan horse* on his machine or hacking into a machine. A Corrupt query directed against a client U may also be used to replace the value of $pw_S[U]$ used by server S. This is the role of the second argument to Corrupt. This capability allows a dishonest client A to try to defeat protocol aims by installing a strange string as a server S's password $pw_S[A]$ [1].

Freshness

There are two notions of freshness presented for this protocol, freshness with and without forward secrecy. These two notions of freshness presented have some events associated with them. $\mathsf{RevealTo}(U, i)$ event is true if there was at some point in time a $Reveal(U, i)$ query. $\mathsf{RevealToPartner}(U, i)$ event is true if there was, at some point in time, a query $Reveal(U', i')$ and $\Pi^{i'}_{U'}$ is partner to Π^i_U. $\mathsf{SomebodyWasCorrupted}$ event is true if at some point in time, a $Corrupt(U'', pw)$ query for some principle U'' and some pw. $\mathsf{SomebodyWasCorruptedBeforeTheTestQuery}$ event is true if there was a query $Corrupt(U', pw)$ and this query was made before $Test(U, i)$. We say that $\mathsf{Manipulated}(U, i)$ event is true if there was at some point in time, a $Send(U, i, M)$ query, for some string M [1].

The basic notion of freshness with no requirement of forward secrecy, defines an instance as *unfresh* if $\mathsf{RevealTo}(U, i)$ query or $\mathsf{RevealToPartnerOf}(U, i)$ queries were made or $\mathsf{SomebodyWasCorrupted}$ is true. In any of the other cases an instance is considered *fresh.*

Freshness with forward secrecy defines an instance as *unfresh* if $\mathsf{RevealTo}(U, i)$ or $\mathsf{RevealToPartnerOf}(U, i)$ or $\mathsf{SomebodyWasCorruptedBeforeTheTestQuery}$ and $\mathsf{Manipulated}(U, i)$.

In any other cases it is considered *fs-fresh*. This definition of security gives credit to the adversary $\mathcal{A}$ if it specifies a fresh (*fs-fresh*) oracle and then correctly identifies if $\mathcal{A}$ is provided with the SK from that oracle or else a random SK. Two cases are presented, according to whether or not *forward secrecy* is expected. For the basic notion of security (*fresh/unfresh*) it is pessimistically assumed that a *Corrupt* query does reveal the session key, so any *Corrupt* query makes all oracles *unfresh*. For the version of the definition with forward secrecy a *Corrupt* query may reveal a SK only if the *Corrupt* query was made before the *Test* query. It is also required that the *Test* query was to an oracle that was the target of a *Send* query. This acts to build a requirement: that even after the *Corrupt* query, session keys exchanged by principals who behave honestly are still *fs-fresh* [1].

Accepting and Terminating

We differentiate between an instance accepting and terminating. When an instance terminates, it means that it has what it wants and won't send out any further messages. An instance may wish to accept now, and terminate later, this happens typically when an instance believes it is holding a good session key, but, before using it, the instance wants confirmation that its communication partner really exists, and is holding the same session key. The instance can accomplish this by accepting now, but waiting for a confirmation message to terminate [1].

Advantage of the Adversary

Let Event *Succ* be the event when adversary $\mathcal{A}$ succeeds. Event *Succ* occurs if the adversary asks for a single *Test-Query* $Test(U, i)$ where Π_U^i has terminated, it is *fresh* and $\mathcal{A}$ outputs a single bit b' which $b' = b$ (where b is the bit selected by the Test query). The advantage of an adversary $\mathcal{A}$ attacking protocol $\mathcal{P}$, is defined as $Adv_{P,\mathcal{A}}^{ake} = 2Pr[Succ] - 1$. Similarly, the **ake-fs** Advantage, $Adv_{P,\mathcal{A}}^{ake}()$. An extra condition is required to the oracle Π_U^i, to which the *Test-Query* is directed, must be **fs-fresh** [1]. If the adversary was unrestricted, success would be trivial (since the adversary could submit a *Reveal* query for the same instance submitted to the *Test* oracle). Some restrictions are imposed to prevent it. A polynomially bounded adversary will be able to break any protocol by attempting to impersonate a user and trying all passwords one-by-one. So, a given protocol is secure when this kind of attack is the best an adversary can do.

In [11], the resources available to an adversary $\mathcal{A}$ were measured and it was concluded that a protocol $\mathcal{P}$ is secure if, when passwords are chosen from a dictionary of size N, the adversary's advantage in attacking the protocol is bounded by $\mathcal{O}(\frac{q_{send}}{N}) + \varepsilon(\lambda)$, where q_{send} is the number of calls the adversary makes to the *Send* oracle, and $\varepsilon(.)$ is some negligible function. The first term represents the fact that the adversary can do no better than guess a password during each call to the *Send* oracle. Even polymonially-many calls to the *Execute* oracle (which are passive observations of valid executions) and the *Reveal* oracle (compromise of short-term session keys) are of no help to an adversary. Only online impersonation attacks (which are harder to mount and easier to detect) give the adversary a non-negligible advantage.

Authentication

In an protocol execution $\mathcal{P}$, an adversary $\mathcal{A}$ violates client-to-server authentication if some *server oracle* terminates but has no *partner oracle*. *c2s advantage* represents the probability of this event, and is denoted by $Adv^{c2s}_{P,\mathcal{A}}$. An adversary violates server-to-client authentication if some client oracle terminates but has no partner oracle. Let *s2c advantage* be the probability of this event, denoted by $Adv^{s2c}_{P,\mathcal{A}}$. Also, an adversary violates mutual authentication if some oracle terminates, but has no partner oracle. *ma-advantage* defines the probability of this event, and is denoted by $Adv^{ma}_{P,\mathcal{A}}$.

An AKE protocol is easily modified to provide authentication. These transformations use a well-known approach of using the distributed session key to construct a simple *authenticator* for the other party [1].

Relation to information theoretic security model

The security models for information theoretically secure Authenticated Key Agreement (AKA) and computationally secure Password-Authenticated Key Exchange (PAKE) that we have studied have fundamental differences that are obvious. The former allows for an unbounded adversary, yet restricts the attack scenario to a single exchange. On the other hand, the latter allows for many interactions, even allowing the adversary to exploit cross-session attacks, yet only considers adversaries that can be modeled as probabilistic polynomial time algorithms.

Nevertheless, the models do have some properties in common. In particular, both definitions identify two types of events in a protocol execution, corresponding to two different types of states that can be reached by participants:

- Getting the values needed for preparing the key. This state is described as *Accepting* in PAKE and `KeyDerived` in AKA.

- Sending of the authenticated ciphertext. The is the final state and is described as *terminating* in PAKE and `Keyconfirmed` in AKA.

However, in the PAKE model, the idea of key confirmation is the following: when one of the participants gets the needed information from the other party and holds a good session key, it will still wait for some type of confirmation that their partner really exists, and is holding the same session key. The participant can accomplish this by *Accepting* now, but waiting for a confirmation message to terminate [1]. In AKA, Alice reaches `KeyDerived` after Bob has issued a challenge and Alice possesses a *random candidate key*. Alice, contrary to what happens in PAKE, has no information if her key is shared with Bob nor if he is even involved in the protocol [7]. When in PAKE, a participant *Terminates*, it means that it has what it wants and won't send out any further messages [1]. In AKA, to Bob reaching the `KeyConfirmed` state and getting a candidate key means that Alice must have been involved in the protocol execution, must have reached the `KeyDerived` state, and the two parties have shared the same key [7].

4.2 A concrete PAKE protocol

Katz, Ostrovsky and Yung present their PAKE protocol presented in [11]. This protocol relies on the following building blocks: an extension of the Cramer-Shoup cryptosystem, secure under adaptive chosen-ciphertext attack; a one-time signature scheme secure against existential forgery; and finally, the proof also relies on the Decisional Diffie-Hellman assumption.

The extended Cramer-Shoup encryption is an extension to the original algorithm. We firstly studied the security analysis of Cramer-Shoup's $\mathsf{CS}_{\mathsf{1A}}$ algorithm [5]. This particular version was the basis for the extension presented. Extended Cramer-Shoup uses some extra arguments, which are used to tag a principal as either a client or a server. The protocol distinguishes between these two types of principals as they have two different roles. Two different encryption schemes are used, and labeled client and server encryption, respectively. We have analyzed where the original and the extension scheme diverged, and focused our study in the security games that were affected. The sole difference in the security proofs between the two schemes concerns the use of the hash function. Contrarily to the $\mathsf{CS}_{\mathsf{1A}}$ scheme, the hash arguments in the extended version are not randomly sampled. Some of the extra arguments that form the hash function argument are given by the adversary. This means that the original security proof by Cramer and Shoup [5] does not apply, and the notion of a UOWHF must be used instead the use of a TCR hash function.

To deal with this, we have constructed an alternative sequence of games, where the security of the extension is proven.

No particular one-time signature scheme was predefined, the only requirement was that the one-time signature must be secure against existential forgery [11]. We opted to study one of the most famous one-time signature schemes: Lamport's One-time signature. Based on the security proof where the signature is proven weakly existentially unforgeable [10], we have created a modified proof that shows this signature is strongly existentially unforgeable.

After each of these security analyses were made, we performed a concrete security study where we found the minimum values needed in each component in order to the protocol reaches a secret key for the two parties that will assure a 128-bit security level.

Protocol description

Consider figure 4.1. Let p, q be primes such that $q|p-1$, and let $\mathcal{G}$ be a subgroup of $\mathbb{Z}_p^*$ of order q in which the Decisional Diffie-Hellman assumption holds. During the initialization phase, generators $g_1, g_2, h, c, d \in \mathcal{G}$ and a function $\mathcal{H}$ from a family of universal one-way hash functions are chosen at random and published. No one must know the discrete logarithm of any of the generators with respect to any other, for this matter either a trusted party who generates the public information or else a source of randomness which can be used to publicly derive the information is needed.

As part of the initialization phase, a password pw_C is chosen randomly for a client. All passwords lie in $\mathbb{Z}_q$, for typical values of $|q|$. This will be a valid assumption for human-memorable passwords.

Execution of the protocol is as follows: When a client wants to connect to a server, the client first runs the key generation algorithm for the one-time signature scheme, giving VK and SK.

Then, the client computes client-encryption of $g_1^{pw_C}$. This, along with the client's identification, is sent to the server as the first message. The server chooses random elements x_2, y_2, z_2, w_2 from $\mathbb{Z}_q$, computes α' using the first message, and forms $g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2}$. The server then computes a server-encryption of $g_1^{pw_C}$. This is sent back to the client as the second message. The client selects random elements x_1, y_1, z_1, w_1 from $\mathbb{Z}_q$, computes β' using the second message, and forms $K = g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\beta'})^{w_1}$. Finally, β' and K are signed using the signing key SK, which was generated in the first step.

Client	public information: p,q,g_1,g_2,h,c,d,$\mathcal{H}$	*Server*
$(\mathsf{VK},\mathsf{SK}) \leftarrow \mathsf{Gen}(1^\lambda)$ $r_1 \leftarrow \mathbb{Z}_q$ $A = g_1^{r_1}; B = g_2^{r1}$ $C = h^{r_1} g_1^{pw_C}$ $\alpha = \mathcal{H}(Client\vert\mathsf{VK}\vert A\vert B\vert C)$ $D = (cd^\alpha)^{r_1}$	$\xrightarrow{Client\vert\mathsf{VK}\vert A\vert B\vert C\vert D}$	
		$x_2, y_2, z_2, w_2, r_2 \leftarrow \mathbb{Z}_q$ $\alpha' = \mathcal{H}(Client\vert\mathsf{VK}\vert A\vert B\vert C)$ $E = g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2}$ $F = g_1^{r_2}; G = g_2^{r_2}$ $I = h^{r_2} g_1^{pw_C}$ $\beta = \mathcal{H}(Server\vert E\vert F\vert G\vert I)$ $J = (cd^\beta)^{r_2}$
	$\xleftarrow{Server\vert E\vert F\vert G\vert I\vert J}$	
$x_1, y_1, z_1, w_1 \leftarrow \mathbb{Z}_q$ $\beta' = \mathcal{H}(Server\vert E\vert F\vert G\vert I)$ $K = g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\beta'})^{w_1}$ $Sig = \mathsf{Sign}(\mathsf{SK}, \beta'\vert K)$	$\xrightarrow{K\vert Sig}$	

		if $\mathsf{Vrfy}(\mathsf{VK}, (\beta\vert K), Sig) = 1$
		$C' = C/g_1^{pw_C}$
		$sk_S = K^{r_2} A^{x_2} B^{y_2} (C')^{z_2} D^{w_2}$
		else $sk_S \leftarrow \mathcal{G}$

$I' = I/g_1^{pw_C}$ $sk_C = E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1}$		

Figure 4.1: protocol for password authentication key exchange [11].

Correctness

In a honest execution of the protocol, client and server calculate identical session keys. To see this, note that $\alpha = \alpha'$ and $\beta = \beta'$ in an honest execution. Then $sk_C = (g_1^{x_2} g_2^{y_2} h^{z_2} (cd^\alpha)^{w_2})^{r_1} g_1^{r_2 x_1} g_2^{r_2 y_1} h^{r_2 z_1} (cd^\beta)^{r_2 w_1}$ and $sk_S = (g_1^{x_1} g_2^{y_1} h^{z_1} (cd^\beta)^{w_1})^{r_2} g_1^{r_1 x_2} g_2^{r_1 y_2} h^{r_1 z_2} (cd^\alpha)^{r_1 w_2}$ is easy to verify that $sk_C = sk_S$ [11].

Security of the scheme

Let P be the protocol in Figure 4.1, where passwords are chosen from a dictionary of size N, and let $\lambda = |q|$ be the security parameter. Let $\mathcal{A}$ be a probabilistic, polynomial-time adversary that asks $q_{execute}$, q_{send} and q_{reveal} queries to the respective oracles. Then the advantage of an adversary $\mathcal{A}$ in attacking this protocol is proven in [11] to be

$$\begin{aligned} Adv_{P,A}^{ake}(\lambda) < \frac{q_{send}}{2N} &+ 2q_{send}\varepsilon_{sig}(\lambda,t) + 2\varepsilon_{ddh}(\lambda,t) + 2q_{send}\varepsilon_{cs}(\lambda,t,q_{send}/2) \\ &+ 2q_{send}\varepsilon_{hash}(\lambda,t) + \frac{min\{2q_{reveal}, q_{send}\}}{q} + \frac{2min\{q_{reveal}, q_{execute}\}}{q^2}. \end{aligned} \tag{4.1}$$

We take a closer look at the negligible terms present in the advantage of an adversary $\mathcal{A}$ against PAKE protocol. There are three types of queries that $\mathcal{A}$ can call. We will set $\mathcal{A}$'s capability to send up to 2^{40} send queries. Usually there are less reveal and execute queries than send queries. Nevertheless we set $q_{reveal} = q_{execute} = q_{send} = 2^{40}$ queries.

4.3 Lamport's one-time signature scheme

A one-time signature is a signature scheme which is, as its name suggests, secure when is used only one time. We describe the building-block functions of Lamport's one-time signature scheme as $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ in Figure 4.2, where Gen is a probabilistic algorithm, Sign, Vrfy are deterministic algorithms and f is a one-way function.

```
$\mathsf{Gen}(1^\lambda)$
For $1 \leq i \leq l$:
    $x_{i,0} \leftarrow \{0,1\}^\lambda, x_{i,1} \leftarrow \{0,1\}^\lambda$,
    $y_{i,0} = f(x_{i,0}), y_{i,1} = f(x_{i,1})$;
$\mathsf{SK} = \{x_{i,b} : 1 \leq i \leq l, b \in \{0,1\}\}$;
$\mathsf{VK} = \{y_{i,b} : 1 \leq i \leq l, b \in \{0,1\}\}$;
return $(\mathsf{VK}, \mathsf{SK})$.

$\mathsf{Sign}(\mathsf{SK}, \mathsf{m})$
$m = (m_1, \ldots, m_l)$;
$\sigma = (x_{1,m_1}, \ldots, x_{l,m_l})$;
return $\sigma$.

$\mathsf{Vrfy}(\mathsf{VK}, \mathsf{m}, \sigma)$
$m = (m_1, \ldots, m_l), \sigma = (x_1, \ldots, x_l)$;
If forall $1 \leq i \leq l$ $f(x_i) = y_{i,m_i}$,
    return 1;
else
    return 0.
```

Figure 4.2: Lamport's one-time signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$.

Security Analysis

Theorem 4. *([10]) Let $l = l(\lambda)$ be any polynomial. If a function f is target collision resistant and one-way, then Lamport's one-time signature scheme is strongly existentially unforgeable against chosen-message attack for messages of length $l(\lambda)$.*

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be Lamport's scheme, consider the experiment in figure 4.2 for a probabilistic, polynomial-time adversary $\mathcal{A}$. We recall the strong unforgeability experiment for the particular case of Π in figure 4.3, denoted as $SigForge^{\mathsf{sEUF-CMA}}_{\mathcal{A},\Pi}(\lambda)$. Consider the particular case when a successful adversary $\mathcal{A}$ that outputs a message already queried to the signing oracle but now with a different signature. We name the event when $\mathcal{A}$ submits a pair (m, σ) with $m = m'$ and $\sigma \neq \sigma'$ as event E.

For legibility purposes, we rename the experiments of strong and weak existential unforgeability, where the adversary *must* forge on a new message as follows:

- $\mathsf{WEAK} = SigForge^{\mathsf{wEUF-CMA}}_{\mathcal{A},\Pi}(\lambda)$;
- $\mathsf{STRONG} = SigForge^{\mathsf{sEUF-CMA}}_{\mathcal{A},\Pi}(\lambda)$.

```
**Experiment** $SigForge^{\mathsf{sEUF-CMA}}_{\mathcal{A},\Pi}(\lambda)$:

$(\mathsf{SK},\mathsf{VK}) \leftarrow \mathsf{Gen}(1^\lambda)$;
$(m,\sigma) \leftarrow \mathcal{A}^{SignOracle(m')}(\mathsf{VK})$;
if $(m,\sigma) \neq (m',\sigma')$, $\mathsf{Vrfy}(\mathsf{VK},\mathsf{m},\sigma) = 1$, return 1;
else return 0.

$\mathsf{Gen}(1^\lambda)$
For $1 \leq i \leq l$:
    $x_{i,0} \leftarrow \{0,1\}^\lambda, x_{i,1} \leftarrow \{0,1\}^\lambda$,
    $y_{i,0} = f(x_{i,0}), y_{i,1} = f(x_{i,1})$;
$\mathsf{SK} = \{x_{i,b} : 1 \leq i \leq l, b \in \{0,1\}\}$;
$\mathsf{VK} = \{y_{i,b} : 1 \leq i \leq l, b \in \{0,1\}\}$;
return $(\mathsf{VK},\mathsf{SK})$.

$\mathsf{Sign}(\mathsf{SK},\mathsf{m})$
$m = (m_1,\ldots,m_l)$;
$\sigma = (x_{1,m_1},\ldots,x_{l,m_l})$;
return $\sigma$.

$\mathsf{Vrfy}(\mathsf{VK},\mathsf{m},\sigma)$
$m = (m_1,\ldots,m_l), \sigma = (x_1,\ldots,x_l)$;
If forall $1 \leq i \leq l$ $f(x_i) = y_{i,m_i}$,
    return 1;
else
    return 0.
```

Figure 4.3: Strong existential unforgeability experiment against a probabilistic, polynomial-time adversary $\mathcal{A}$.

Taking in consideration the occurrence of event E, we represent $\mathcal{A}$'s success probability in $SigForge^{\mathsf{sEUF-CMA}}_{\mathcal{A},\Pi}(\lambda)$ as:

$$\begin{aligned} Pr[\mathsf{STRONG}=1] &= Pr[\mathsf{STRONG}=1 \wedge \mathsf{E}] + Pr[\mathsf{STRONG}=1 \wedge \neg\mathsf{E}]; && (4.2)\\ &= Pr[\mathsf{E}]Pr[\mathsf{STRONG}=1|\mathsf{E}] + Pr[\neg\mathsf{E}]Pr[\mathsf{STRONG}=1|\neg\mathsf{E}]. && (4.3)\end{aligned}$$

Event E denotes an event that is not considered in the weak unforgeability experiment $SigForge^{\mathsf{wEUF-CMA}}_{\mathcal{A},\Pi}(\lambda)$: reusing the signature previously issued by an adversary. We can then assume that the advantage of $\mathcal{A}$ in $SigForge^{\mathsf{sEUF-CMA}}_{\mathcal{A},\Pi}(\lambda)$ if event E does not occur, is equal to the advantage of $\mathcal{A}$ in $SigForge^{\mathsf{wEUF-CMA}}_{\mathcal{A},\Pi}(\lambda)$,

$$Pr[\mathsf{STRONG}=1|\neg\mathsf{E}] = Pr[\mathsf{WEAK}=1]. \tag{4.4}$$

So $Pr[\mathsf{STRONG}=1]$ can be described as:

$$Pr[\mathsf{STRONG}=1] \leq Pr[\mathsf{E}]Pr[\mathsf{STRONG}=1|\mathsf{E}] + Pr[\mathsf{WEAK}=1]. \tag{4.5}$$

The proof now follows from the following two claims, the first of which is proven by Katz and Lindell in [10].

Claim 1. $Adv^{\mathsf{wEUF-CMA}}_{A,\Pi}(\lambda) = Pr[SigForge^{\mathsf{wEUF-CMA}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq 2l(\lambda)\varepsilon_{owf}(\lambda)$**.**

Claim 2. $Pr[Sigforge^{\mathsf{sEUF-CMA}}_{A,\Pi}(\lambda) = 1|\mathsf{E}] \leq 2l(\lambda)\varepsilon_{tcr}(\lambda)$**.**

Proof. The intuition is the following: if event E occurs, the adversary $\mathcal{A}$ has successfully found a new preimage to m' under f, which is the same as $\mathcal{A}$ breaking the target collision resistance property of f.

```
Algorithm $\mathcal{B}(x^*, y^*)$
$(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(x^*, y^*)$;
$(m, \sigma) \leftarrow \mathcal{A}^{signOracle(m')}(\mathsf{VK})$;
if $m \neq m'$ halt; if $\sigma = \sigma'$ halt;
if $\mathsf{Vrfy}(\mathsf{VK}, m, \sigma) = 1$, return $x_{i^*}$;
else halt.
```

```
$\mathsf{Gen}(x^*, y^*)$
$i^* \leftarrow \{1, \dots, l\}, b^* \leftarrow \{0, 1\}$;
$y_{i^*,b^*} = y^*; x_{i^*,b^*} = x^*$;
forall $1 \leq i \leq l, b \in \{0, 1\}$ :
    if $(i, b) \neq (i^*, b^*)$,
        $x_{i,b} \leftarrow \{0, 1\}^\lambda$, $y_{i,b} = f(x_{i,b})$;
return $(y, x)$.
```

```
$\mathsf{Sign}(\mathsf{SK}, m)$
if $m_{i^*} \neq b^*$ halt;
$\sigma = (x_{1,m_1}, \dots, x_{l,m_l})$;
return $\sigma$.
```

Figure 4.4: Algorithm $\mathcal{B}$.

Formally, we have created algorithm $\mathcal{B}$, presented in figure 4.4, that given $y^*, x^* \in \{0,1\}^\lambda$ uses $\mathcal{A}$ to break the TCR assumption. We describe the advantage of breaking the TCR assumption as $\varepsilon_{tcr}(\lambda)$ which is assumed to be negligible, and so we must have that B' probability of success is upper bounded by $\varepsilon_{tcr}(\lambda)$.

A marked bit (i^*, b^*) used in the message proposed is independently chosen from $\mathcal{A}$'s view. The probability of x^* being used in a signature is 1/2. Adversary $\mathcal{A}$ needs to produce a signature σ where $\sigma \neq \sigma'$. A signature σ is different from σ' if has at least one element where $\sigma_i \neq \sigma'_i$. The probability of $\mathcal{A}$ altering the marked bit σ_{i^*} is $1/l$. So the probability of $\mathcal{A}$ choosing a marked x^* in SK is $\frac{1}{2l}$.

$$\frac{1}{2l(\lambda)} Pr[\mathsf{STRONG} = 1 | \mathsf{E}] \leq \varepsilon_{tcr}(\lambda); \tag{4.6}$$

$$Pr[\mathsf{STRONG} = 1 | \mathsf{E}] \leq 2l(\lambda)\varepsilon_{tcr}(\lambda). \tag{4.7}$$

□

Finally we calculate the overall advantage of $\mathcal{A}$ in the Strong unforgeability experiment, as follows.

$$Pr[\mathsf{STRONG} = 1] \leq Pr[\mathsf{WEAK} = 1] + Pr[\mathsf{STRONG} = 1 | \mathsf{E}]; \tag{4.8}$$

$$Pr[\mathsf{STRONG} = 1] \leq 2l(\lambda)\varepsilon_{tcr}(\lambda) + 2l(\lambda)\varepsilon_{owf}(\lambda); \tag{4.9}$$

4.4 Extended Cramer-Shoup cryptosystem

The Cramer-Shoup cryptosystem is an encryption scheme secure under adaptive chosen ciphertext attack [5]. An extension to this cryptosystem was presented in [11] which has two different encryption versions, where both versions use the same public parameters. This extension is also claimed secure under adaptive chosen ciphertext attacks. We will distinguish and present with detail the two versions of the extension as Π_C regarding the client encryption and Π_S regarding the server encryption. We will then perform a security analysis on both versions, based on the security analysis of the $\mathsf{CS_{1A}}$ version of the Cramer-Shoup cryptosystem [5]. In Figure 4.5 we present these extended schemes in detail. Keeping in mind the notation and numbering used in similar diagrams in [5], we denote the extra steps needed on this extension as X^* (*e.g.* $E1^*$).

```
$\mathsf{Gen}_C(\Gamma[\hat{G}, G, g, q])$
$w \xleftarrow{R} \mathbb{Z}_q^*$; $x_1, x_2, y_1, y_2, z \xleftarrow{R} \mathbb{Z}_q$;
$hk \xleftarrow{R} \mathsf{HF.Keyspace}_\Gamma$;
$\mathsf{PK} := (\Gamma[\hat{G}, G, g, q], hk, \hat{g}, e, f, h)$,
$\mathsf{PK} \in [S] \times [\mathsf{HF.Keyspace}] \times G^4$,
where $\hat{g} = g^w$, $e = g^{x_1}\hat{g}^{x_2}$, $f = g^{y_1}\hat{g}^{y_2}$, $h = g^z$;
$\mathsf{SK} := (\Gamma[\hat{G}, G, g, q], hk, x_1, x_2, y_1, y_2, z)$,
$\mathsf{SK} \in [S] \times [\mathsf{HF.Keyspace}] \times \mathbb{Z}_q^5$;
return $(\mathsf{PK}, \mathsf{SK})$.
```

```
$\mathsf{Enc}_C(m, Client, \mathsf{PK})$
$E1^*$ : $(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{SigGen}()$;
$E2^*$ : $A \leftarrow Client$;
$E3^*$ : $B \leftarrow \mathsf{VK}$;
$E1$ : $u \xleftarrow{R} \mathbb{Z}_q$;
$E2$ : $a \leftarrow g^u$;
$E3$ : $\hat{a} \leftarrow \hat{g}^u$;
$E4$ : $b \leftarrow h^u$;
$E5$ : $c \leftarrow b.m$;
$E6$ : $v \leftarrow \mathsf{HF}^\Gamma_{hk}(A, B, a, \hat{a}, c)$;
$E7$ : $d \leftarrow e^u f^{uv}$;
return $(\psi = \langle A, B, a, \hat{a}, c, d\rangle, \mathsf{SK})$.
```

```
$\mathsf{Dec}_C(\psi := \langle A, B, a, \hat{a}, c, d\rangle)$
$D1$ : parse $\psi$ as a 6-tuple $(A, B, a, \hat{a}, c, d) \in Client \times \hat{G}^5$, output reject and halt if it is not of this form;
$D2$ : if $A \in Client$ and $B$, $a$, $\hat{a}$, $c \in G$, otherwise output reject and halt;
$D3$ : $v \leftarrow \mathsf{HF}^\Gamma_{hk}(A, B, a, \hat{a}, c)$;
$D4$ : if $d = a^{x_1+y_1v}\hat{a}^{x_2+y_2v}$, otherwise output reject and halt;
$D5$ : $b \leftarrow a^z$;
$D6$ : $m \leftarrow c \cdot b^{-1}$;
return $m$.
```

```
$\mathsf{Gen}_S(\Gamma[\hat{G}, G, g, q])$
$w \xleftarrow{R} \mathbb{Z}_q^*$; $x_1, x_2, y_1, y_2, z \xleftarrow{R} \mathbb{Z}_q$;
$hk \xleftarrow{R} \mathsf{HF.Keyspace}_\Gamma$;
$\mathsf{PK} := (\Gamma[\hat{G}, G, g, q], hk, \hat{g}, e, f, h)$,
$\mathsf{PK} \in [S] \times [\mathsf{HF.Keyspace}] \times G^4$,
where $\hat{g} = g^w$, $e = g^{x_1}\hat{g}^{x_2}$, $f = g^{y_1}\hat{g}^{y_2}$, $h = g^z$;
$\mathsf{SK} := (\Gamma[\hat{G}, G, g, q], hk, x_1, x_2, y_1, y_2, z)$,
$\mathsf{SK} \in [S] \times [\mathsf{HF.Keyspace}] \times \mathbb{Z}_q^5$;
return $(\mathsf{PK}, \mathsf{SK})$.
```

```
$\mathsf{Enc}_S(m, Server, \alpha, \mathsf{PK})$
$E1^*$ : $x', y', z', l \leftarrow \mathbb{Z}_q$;
$E2^*$ : $A \leftarrow Server$;
$E3^*$ : $B \leftarrow g^{x'}\hat{g}^{y'}h^{z'}(cd^\alpha)^l$;
$E1$ : $u \xleftarrow{R} \mathbb{Z}_q$;
$E2$ : $a \leftarrow g^u$;
$E3$ : $\hat{a} \leftarrow \hat{g}^u$;
$E4$ : $b \leftarrow h^u$;
$E5$ : $c \leftarrow b.m$;
$E6$ : $v \leftarrow \mathsf{HF}^\Gamma_{hk}(A, B, a, \hat{a}, c)$;
$E7$ : $d \leftarrow e^u f^{uv}$;
return $(\psi = \langle A, B, a, \hat{a}, c, d\rangle, x', y', z', l)$.
```

```
$\mathsf{Dec}_S(\psi := \langle A, B, a, \hat{a}, c, d\rangle)$
$D1$ : parse $\psi$ as a 6-tuple $(A, B, a, \hat{a}, c, d) \in Server \times \hat{G}^5$, output reject and halt if it is not of this form;
$D2$ : if $A \in Server$ and $B$, $a$, $\hat{a}$, $c \in G$, otherwise output reject and halt;
$D3$ : $v \leftarrow \mathsf{HF}^\Gamma_{hk}(A, B, a, \hat{a}, c)$;
$D4$ : if $d = a^{x_1+y_1v}\hat{a}^{x_2+y_2v}$, otherwise output reject and halt;
$D5$ : $b \leftarrow a^z$;
$D6$ : $m \leftarrow c \cdot b^{-1}$;
return $m$.
```

Figure 4.5: Extended Cramer-Shoup encryption scheme. In the left side we present the client version $\Pi_C = (\mathsf{Gen}_C, \mathsf{Enc}_C, \mathsf{Dec}_C)$, and in the right side the server version $\Pi_S = (\mathsf{Gen}_S, \mathsf{Enc}_S, \mathsf{Dec}_S)$.

Security model

The definitions of security that must be met by these encryption schemes are very close to the standard notion of IND-CCA. The two security experiments are described in Figure 4.6. Compared to an IND-CCA experiment, these experiments only differ in step 2., where $\mathcal{A}$ ($\mathcal{A}'$) gives only one message m^*. Although, like in the IND-CCA experiment, the adversary must be able to distinguish between two messages, in this case, the adversary must choose between message m^* and a randomly created message. Furthermore, the adversary gets to choose additional parameters including the identity of the client/server.

$\underline{Exp_{\Pi_C,\mathcal{A}}}$	$\underline{Exp_{\Pi_S,\mathcal{A}'}}$
• $(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{Gen}_C(\Gamma)$;	• $(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{Gen}_S(\Gamma)$;
• $(m^*, C, st) \leftarrow \mathcal{A}_1^{\mathsf{Decrypt}(.)}(\mathsf{PK})$;	• $(m^*, S, \alpha, st) \leftarrow \mathcal{A}'^{\mathsf{Decrypt}(.)}_1(\mathsf{PK})$;
• $m_0 \overset{R}{\leftarrow} \mathsf{Msg.Keyspace}$; $m_1 = m^*$; $b \overset{R}{\leftarrow} \{0,1\}$;	• $m_0 \overset{R}{\leftarrow} \mathsf{Msg.Keyspace}$; $m_1 = m^*$; $b \overset{R}{\leftarrow} \{0,1\}$;
• $c \leftarrow \mathsf{Enc}_C(m_b, C, \mathsf{PK})$;	• $c \leftarrow \mathsf{Enc}_S(m_b, S, \alpha, \mathsf{PK})$;
• $b' \overset{R}{\leftarrow} \mathcal{A}_2^{\mathsf{Decrypt}(.)}(c, st)$;	• $b' \overset{R}{\leftarrow} \mathcal{A}'^{\mathsf{Decrypt}(.)}_2(c, st)$;
• $\mathsf{return}(b = b')$.	• $\mathsf{return}(b = b')$.

Figure 4.6: Security experiments for the client version Π_C (reps. server version Π_S) of extended Cramer-Shoup.

Security analysis

The Cramer-Shoup cryptosystem ($\mathsf{CS_{1A}}$ version) is secure under adaptive chosen ciphertext attack if decisional Diffie-Hellman holds in $\mathcal{G}$ and target collision resistance assumption holds for an hash function HF [5]. The advantage of a polynomially-bounded adversary $\mathcal{A}$ against an adaptive chosen ciphertext attack is given by

$$\mathsf{AdvCCA}_{\mathsf{CS1},\mathcal{A}}(\lambda|\Gamma) \leq \mathsf{AdvDDH}_{\mathcal{G},\mathcal{A}_1}(\lambda|\Gamma) + \mathsf{AdvTCR}_{\mathsf{HF},\mathcal{A}_2}(\lambda|\Gamma) + (Q_{\mathcal{A}}(\lambda)+4)/q. \quad (4.10)$$

We have studied the sequence of security games regarding $\mathsf{CS_{1A}}$, and looked at the proof steps in the analysis of the $\mathsf{CS_{1A}}$ that did not apply to the analysis of the extended version. Concretely, we have considered the security games $G_1, \ldots, G_4$ from both client and server versions of the extension to be equivalent to those used in the proof of $\mathsf{CS_{1A}}$. Beyond the extra parameters that are computed in the extension, the only difference relates to the hash function HF. In the extension HF has extra parameters, some of them are not randomly sampled, but given by the adversary. We therefore have looked at the parts of the security proof that are affected by this change, in order to have a valid security reduction for extended Cramer-Shoup. The critical point is the game hop leading to game G_5, based on a reduction to the TCR property of hash function HF. This game includes a new rule defined as:

> *if the adversary sends a* $\psi = (a, \hat{a}, c) \neq (a^*, \hat{a}^*, c^*)$ *and* $\mathsf{HF}(\psi) = \mathsf{HF}(\psi^*)$, *then the decryption oracle outputs reject.*

In the extension the rejection rule must include (in both versions) the extra variables A and B, where A is chosen by the adversary. So the rejection rule becomes:

> *if the adversary sends a* $\psi = (A, B, a, \hat{a}, c) \neq (A^*, B^*, a^*, \hat{a}^*, c^*)$ *and* $\mathsf{HF}(\psi) = \mathsf{HF}(\psi^*)$*, then the decryption oracle outputs reject.*

Since part of the hash function input is provided by the adversary, we must assume that the HF is a *universal one-way hash function*, instead of TCR as defined in the original paper.

We show the modified games G_4 and G_5 for both client and server versions in detail below. As the rejection rule only alters the games in the Decryption Oracle, we will keep the description of both the games in common with exception to this component. Keeping in mind the notation used in [5], we denote the extra steps needed in this extension as X^* (*e.g.* $E1^*$) and the altered steps in the previous security games as X' (*e.g.* $E3'$).

The two games for the client version Π_C are presented in Figure 4.7 where we present on top the unaltered functions for the two games, and below two decryption oracles respectively for G_4 and G_5. We marked the altered step in both games underlining it. When compared with the original $\mathsf{CS_{1A}}$ games, the client version of the extension has three extra values VK, $\mathsf{SK} \in \mathbb{Z}_p$ that are calculated independently from the adversary's view. Value $\mathsf{VK} = B$ is used as part of the argument of the hash function HF. Enc_C outputs the pair (ψ, SK) where VK is part of $\psi = \langle A, B, a, \hat{a}, c, d \rangle$. On the other hand, value $A \in \{Client\}$ is given by the adversary to Enc_C and used as part of the argument of the hash function HF.

Figure 4.8 shows the same two games for the server version Π_S. As before, on the top we present the unaltered functions to both games, and below the two decryption oracles for G_4 and G_5 with the altered step underlined. As seen before in the client version, some extra parameters are used that are chosen by the adversary. In this case, two extra parameters are given by the adversary $A \in \{Server\}$ and $\alpha \in \mathbb{Z}_p$. These two extra values are given as arguments to Enc_S. Enc_S returns $\psi = \langle A, B, a, \hat{a}, c, d \rangle$ along with a set of values $\{x', y', z', l\}$, this set of values along with value α and the public parameters are used to create value B, part of ψ.

```
$\mathsf{G}_4$ and $\mathsf{G}_5$ games for $\Pi_C$
$(\mathsf{PK}, \mathsf{SK}) \leftarrow Gen_C(\Gamma)$;
$(m^*, C, st) \leftarrow \mathcal{A}_1^{\mathsf{DecOracle}}(\mathsf{PK})$;
$\psi \leftarrow RealOrRandom(m^*, C)$;
$\sigma' \leftarrow \mathcal{A}_2^{\mathsf{DecOracle}}(\psi, st)$;
$\mathsf{return}(\sigma \overset{?}{=} \sigma')$.
```

```
$Gen_C(\Gamma[\hat{G}, G, g, q])$
$w \xleftarrow{R} \mathbb{Z}_q^*$; $x_1, x_2, y_1, y_2, z \xleftarrow{R} \mathbb{Z}_q$;
$hk \xleftarrow{R} \mathsf{HF.Keyspace}_\Gamma$;
$\mathsf{PK} := (\Gamma[\hat{G}, G, g, q], hk, \hat{g}, e, f, h)$,
$\mathsf{PK} \in [S] \times [\mathsf{HF.Keyspace}] \times G^4$,
where $\hat{g} = g^w$, $e = g^{x_1}\hat{g}^{x_2}$, $f = g^{y_1}\hat{g}^{y_2}$, $h = g^z$;
$\mathsf{SK} := (\Gamma[\hat{G}, G, g, q], hk, x_1, x_2, y_1, y_2, z)$,
$\mathsf{SK} \in [S] \times [\mathsf{HF.Keyspace}] \times \mathbb{Z}_q^5$;
$\mathsf{return}(\mathsf{PK}, \mathsf{SK})$.
```

```
$\mathsf{RealOrRandom}(m^*, C)$
$\sigma \xleftarrow{R} \{0, 1\}$;
if $\sigma = 0$: $m = m^*$;
otherwise,
if $\sigma = 1$: $m \xleftarrow{R} \mathsf{PKE.MsgSpace}$;
$\psi \leftarrow \mathsf{Encryption}(m, C)$;
$\mathsf{return}(\psi)$.
```

```
$Encryption(m, C)$
$E1^*$ : $(\mathsf{VK}, \mathsf{SK}) \leftarrow SigGen()$;
$E2^*$ : $A \leftarrow C$;
$E3^*$ : $B \leftarrow \mathsf{VK}$;
$E1$ : $u \xleftarrow{\mathrm{R}} \mathbb{Z}_q$;
$E2$ : $a \leftarrow g^u$;
$E3'$ : $\hat{u} \xleftarrow{\mathrm{R}} \mathbb{Z}_q \backslash \{u\}; \hat{a} \leftarrow \hat{g}^{\hat{u}}$;
$E4'$ : $b \leftarrow a^{z_1}\hat{a}^{z_2}$;
$E5'$ : $r \xleftarrow{\mathrm{R}} \mathbb{Z}_q; c \leftarrow g^r$;
$E6$ : $v \leftarrow HF_{hk}^\Gamma(A, B, a, \hat{a}, c)$;
$E7'$ : $d \leftarrow a^{x_1+y_1v}\hat{a}^{x_2+y_2v}$;
$\mathsf{return}\ (\psi = \langle A, B, a, \hat{a}, c, d\rangle, \mathsf{SK})$.
```

```
$\mathsf{G}_4$

$\mathsf{DecryptionOracle}(\psi := (A, B, a, \hat{a}, c, d))$
$D1$ : parse $\psi$ as a 5-tuple $(A, B, a, \hat{a}, c, d) \in User \times \hat{G}^5$;
output reject and halt if it is not of this form;
$D2$ : if $A \in User$ and $B$, $a$, $\hat{a}$, $c \in G$;
otherwise return reject and halt;
$D3$ : $v \leftarrow HF_{hk}^\Gamma(A, B, a, \hat{a}, c)$;
$D4'$ : Tests if $\hat{a} = a^w$ e $d = a^{x+yv}$;[1]
return reject and halt if it is not the case.
$D5'$ : $b \leftarrow a^z$;
$D6$ : $m \leftarrow c \cdot b^{-1}$;
$\mathsf{return(m)}$.
```

```
$\mathsf{G}_5$

$\mathsf{DecryptionOracle}(\psi := (A, B, a, \hat{a}, c, d))$
$D1$ : parse $\psi$ as a 5-tuple $(A, B, a, \hat{a}, c, d) \in User \times \hat{G}^5$;
output reject and halt if it is not of this form;
$D2$ : if $A \in User$ and $B$, $a$, $\hat{a}$, $c \in G$;
otherwise return reject and halt;
$D3'$ : if $(A, B, a, \hat{a}, c) \neq (A^*, B^*, a^*, \hat{a}^*, c^*)$
and $v = v^*$ where $v \leftarrow HF_{hk}^\Gamma(A, B, a, \hat{a}, c)$
return reject and halt.
$D4'$ : Tests if $\hat{a} = a^w$ e $d = a^{x+yv}$;[1]
return reject and halt if it is not the case.
$D5'$ : $b \leftarrow a^z$;
$D6$ : $m \leftarrow c \cdot b^{-1}$;
$\mathsf{return(m)}$.
```

[1] $x := x_1 + x_2w$, $y := y_1 + y_2w$, $w := log_g\hat{g}$

Figure 4.7: Security games 4 and 5, regarding the Client version of the extension.

We have created an algorithm B_C which uses $\mathcal{A}$ as a subroutine, and attach the UOWHF property of the hash function if $\mathcal{A}$ can distinguish the game hop in the client version. Similarly, we have created an algorithm B_S that uses $\mathcal{A}'$ as a subroutine, and attach the UOWHF property of the hash function if $\mathcal{A}'$ can distinguish the game hop in the server version. These algorithms are presented in Figure 4.9 and 4.10, respectively.

```
$\mathsf{G}_4$ and $\mathsf{G}_5$ games for $\Pi_S$
$(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Gen}_S(\Gamma)$;
$(m^*, S, \alpha^*, st) \leftarrow \mathcal{A}_1^{\mathsf{DecOracle}}(\mathsf{PK})$;
$\psi \leftarrow RealOrRandom(m^*, S, \alpha^*)$;
$\sigma' \leftarrow \mathcal{A}_2^{\mathsf{DecOracle}}(\psi, st)$;
return$(\sigma \overset{?}{=} \sigma')$.
```

```
RealOrRandom$(\mathsf{m}^*, \mathsf{S}, \alpha^*)$
$\sigma \xleftarrow{R} \{0,1\}$;
if $\sigma = 0$:  $m = m^*$;
otherwise,
if $\sigma = 1$:  $m \xleftarrow{R}$ PKE.MsgSpace;
$\psi \leftarrow Encryption(m, S, \alpha^*)$;
return$(\psi)$.
```

```
$Gen_S(\Gamma[\hat{G}, G, g, q])$
$l \xleftarrow{R} \mathbb{Z}_q^*$; $x_1, x_2, y_1, y_2, z \xleftarrow{R} \mathbb{Z}_q$;
$hk \xleftarrow{R}$ HF.Keyspace$_\Gamma$;
$\mathsf{PK} := (\Gamma[\hat{G}, G, g, q], hk, \hat{g}, e, f, h)$,
$\mathsf{PK} \in [S] \times [\mathsf{HF.Keyspace}] \times G^4$,
where $\hat{g} = g^w$, $e = g^{x_1}\hat{g}^{x_2}$, $f = g^{y_1}\hat{g}^{y_2}$, $h = g^z$;
$\mathsf{SK} := (\Gamma[\hat{G}, G, g, q], hk, x_1, x_2, y_1, y_2, z)$,
$\mathsf{SK} \in [S] \times [\mathsf{HF.Keyspace}] \times \mathbb{Z}_q^5$;
return(PK, SK).
```

```
Encryption$(m, S, \alpha)$
$E1^* : x', y', z', l \leftarrow \mathbb{Z}_q$
$E2^* : A \leftarrow S$;
$E3^* : B \leftarrow g^{x'}\hat{g}^{y'}h^{z'}(ef^\alpha)^l$
$E1 : u \xleftarrow{\mathrm{R}} \mathbb{Z}_q$;
$E2 : a \leftarrow g^u$;
$E3' : \hat{u} \xleftarrow{\mathrm{R}} \mathbb{Z}_q \backslash \{u\}; \hat{a} \leftarrow \hat{g}^{\hat{u}}$;
$E4' : b \leftarrow a^z$;
$E5' : r \xleftarrow{\mathrm{R}} \mathbb{Z}_q; c \leftarrow g^r$;
$E6 : v \leftarrow HF_{hk}^\Gamma(A, B, a, \hat{a}, c)$;
$E7' : d \leftarrow a^{x_1+y_1v}\hat{a}^{x_2+y_2v}$;
return$(\psi = \langle A, B, a, \hat{a}, c, d\rangle, x', y', z', l)$.
```

```
$\mathsf{G}_4$

DecryptionOracle$(\psi := (A, B, a, \hat{a}, c, d))$
$D1$ : parse $\psi$ as a 5-tuple $(A, B, a, \hat{a}, c, d) \in User \times \hat{G}^5$;
output reject and halt if it is not of this form;
$D2$ : if $A \in User$ and $B$, $a$, $\hat{a}$, $c \in G$,
otherwise return reject and halt;
$D3 : v \leftarrow HF_{hk}^\Gamma(A, B, a, \hat{a}, c)$;
$D4'$ : Tests if $\hat{a} = a^w$ e $d = a^{x+yv}$;[1]
return reject and halt if it is not the case;
$D5' : b \leftarrow a^z$;
$D6 : m \leftarrow c \cdot b^{-1}$;
return(m).
```

```
$\mathsf{G}_5$

DecryptionOracle$(\psi := (A, B, a, \hat{a}, c, d))$
$D1$ : parse $\psi$ as a 5-tuple $(A, B, a, \hat{a}, c, d) \in User \times \hat{G}^5$;
output reject and halt if it is not of this form;
$D2$ : if $A \in User$ and $B$, $a$, $\hat{a}$, $c \in G$,
otherwise return reject and halt;
$D3'$ : if $(A, B, a, \hat{a}, c) \neq (A^*, B^*, a^*, \hat{a}^*, c^*)$
and $v = v^*$ where $v \leftarrow HF_{hk}^\Gamma(A, B, a, \hat{a}, c)$
return reject and halt;
$D4'$ : Tests if $\hat{a} = a^w$ e $d = a^{x+yv}$;[1]
return reject and halt if it is not the case;
$D5' : b \leftarrow a^z$;
$D6 : m \leftarrow c \cdot b^{-1}$;
return(m).
```

[1] $x := x_1 + x_2w$, $y := y_1 + y_2w$, $w := log_g\hat{g}$

Figure 4.8: Security games 4 and 5, regarding the Server version of the Extension.

```
$B_C$ Algorithm
$\rho^*, st \leftarrow \mathsf{B}_{\mathsf{C1}}(\Gamma)$;
$hk \xleftarrow{R} \mathsf{HF.Keyspace}_{\Gamma}$;
$\rho' \leftarrow \mathsf{B}_{\mathsf{C2}}(hk, \Gamma, \rho^*, st)$.

$\mathsf{B}_{\mathsf{C2}}(hk, \Gamma, \rho^*, st)$
$(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(\Gamma, hk)$;
$(m', C) \leftarrow \mathcal{A}_1^{DecOracle(\psi)}(\mathsf{PK})$;
$\psi^* \leftarrow \mathsf{RealOrRandom}(m', C)$;
$\rho' \leftarrow \mathcal{A}_2^{decOracle(\psi)}(\psi^*), \psi^* \neq \psi$;
return$(\rho')$.

$\mathsf{Gen}(\Gamma[\hat{G}, G, g, q], hk)$
$x_1, x_2, y_1, y_2, z \xleftarrow{R} \mathbb{Z}_q$;
$\mathsf{PK} := (\Gamma[\hat{G}, G, g, q], hk, \hat{g}, e, f, h)$,
$\mathsf{PK} \in [S] \times [\mathsf{HF.Keyspace}] \times G^4$;
where $e = g^{x_1}\hat{g}^{x_2}$, $f = g^{y_1}\hat{g}^{y_2}$, $h = g^z$;
$\mathsf{SK} := (\Gamma[\hat{G}, G, g, q], hk, x_1, x_2, y_1, y_2, z)$,
$\mathsf{SK} \in [S] \times [\mathsf{HF.Keyspace}] \times \mathbb{Z}_q^5$;
return$(\mathsf{PK}, \mathsf{SK})$.

$\mathsf{DecryptionOracle}(\psi := (A, B, a, \hat{a}, c, d))$
$D1$ : parse $\psi$ as a 5-tuple $(A, B, a, \hat{a}, c, d) \in Server \times \hat{G}^5$,
return reject and **halt** if it is not of this form;
$D2$ : if $A \in Server$ and if $B$, $a$, $\hat{a}$, $c \in G$,
otherwise return reject and **halt**;
$D3'$ : if $(A, B, a, \hat{a}, c) \neq (A^*, B^*, a^*, \hat{a}^*, c^*)$ and $v = v^*$,
where $v \leftarrow \mathsf{HF}_{hk}^{\Gamma}(A, B, a, \hat{a}, c)$ then return$(A, B, a, \hat{a}, c)$ and **halt**,
otherwise return reject and **halt**.
```

```
$\mathsf{B}_{\mathsf{C1}}(\Gamma[\hat{G}, G, g, q])$
$A^* \xleftarrow{R} Client$;
$(\mathsf{VK}^*, \mathsf{SK}^*) \leftarrow \mathsf{SigGen}()$;
$B^* = \mathsf{VK}^*$;
$u \xleftarrow{R} \mathbb{Z}_q; a^* = g^u$;
$w \xleftarrow{R} \mathbb{Z}_q^*; \hat{g} = g^w$;
$\hat{u} \xleftarrow{R} \mathbb{Z}_q/\{u\}; \hat{a}^* = \hat{g}^{\hat{u}}$;
$r \xleftarrow{R} \mathbb{Z}_q; c^* = g^r$;
$\rho^* = (A^*, B^*, a^*, \hat{a}^*, c^*)$;
return$(\rho^*, st := (\phi, u, w, \hat{u}, r))$.

$\mathsf{RealOrRandom}(m, C)$
if $C \neq A^*$, $\mathsf{B}_{\mathsf{C2}}$ **halt**;
otherwise:
$\psi \leftarrow \mathsf{Encryption}(m, C)$;
return$(\psi)$.

$\mathsf{Encryption}(m, C)$
$E6 : v \leftarrow \mathsf{HF}_{hk}^{\Gamma}(A, B, a, \hat{a}, c)$;
$E7' : d \leftarrow a^{x_1+y_1v}\hat{a}^{x_2+y_2v}$;
return$(\psi = \langle A, B, a, \hat{a}, c, d\rangle, \mathsf{SK})$.
```

Figure 4.9: B_C algorithm regarding the client version Π_C.

The arguments for the client and server versions are roughly the same, so we present a single argument referring to an algorithm B. We consider the probability of B winning the game and define two events using the notation present in [5].

- *Event* U_5 - when the value of $A^* \in User$, is equal to a given value by $\mathcal{A}$;
- *Event* C_5 - the new rejection rule in the decryption oracle is activated.

We define the probability of B winning the game subject to the occurrence of event U_5 when the adversary gives an equal value A to the one issued by the algorithm.

$$Pr[\mathsf{B}win] = Pr[\mathsf{B}win \wedge U_5] + Pr[\mathsf{B}win \wedge \neg U_5]; \tag{4.11}$$

$$Pr[\mathsf{B}win] = Pr[\mathsf{B}win|U_5].Pr[U_5] + Pr[\mathsf{B}win|\neg U_5].Pr[\neg U_5]; \tag{4.12}$$

$$Pr[\mathsf{B}win] = Pr[\mathsf{B}win|U_5].\frac{1}{|User|} + 0;^{1} \tag{4.13}$$

$$Pr[\mathsf{B}win] = Pr[\mathsf{B}win|U_5]\frac{1}{|User|}. \tag{4.14}$$

We have stated before that the probability of $\mathcal{A}$ winning B is the same as the

[1] We have that $Pr[\mathsf{B}win|\neg U_5].Pr[\neg U_5] = 0$ because B aborts in the case that the value given

```
$B_S$ Algorithm
$\rho*, st \leftarrow \mathsf{B}_{\mathsf{S1}}(\Gamma)$;
$hk \xleftarrow{R} \mathsf{HF.Keyspace}_\Gamma$;
$\rho' \leftarrow \mathsf{B}_{\mathsf{S2}}(hk, \Gamma, \rho^*, st)$.

$\mathsf{B}_{\mathsf{S2}}(hk, \Gamma, \rho^*, st)$
$\mathsf{PK}, \mathsf{SK} \leftarrow \mathsf{Gen}(\Gamma, hk)$;
$(m', S, \alpha) \leftarrow \mathcal{A}'^{DecOracle(\psi)}_1(\mathsf{PK})$;
$\psi^* \leftarrow \mathsf{RealOrRandom}(m', S, \alpha)$;
$\rho' \leftarrow \mathcal{A}'^{decOracle(\psi)}_2(\psi^*), \psi^* \neq \psi$;
$\mathsf{return}(\rho')$.

$\mathsf{Gen}(\Gamma[\hat{G}, G, g, q], hk)$
$x_1, x_2, y_1, y_2, z \xleftarrow{R} \mathbb{Z}_q$;
$\mathsf{PK} := (\Gamma[\hat{G}, G, g, q], hk, \hat{g}, e, f, h)$,
$\mathsf{PK} \in [S] \times [\mathsf{HF.Keyspace}] \times G^4$ ,
where $e = g^{x_1}\hat{g}^{x_2}$, $f = g^{y_1}\hat{g}^{y_2}$, $h = g^z$;
$\mathsf{SK} := (\Gamma[\hat{G}, G, g, q], hk, x_1, x_2, y_1, y_2, z)$,
$\mathsf{SK} \in [S] \times [\mathsf{HF.Keyspace}] \times \mathbb{Z}_q^5$;
$\mathsf{return}(\mathsf{PK}, \mathsf{SK})$.

$\mathsf{Encryption}(m, S, \alpha)$
$E1^* : y', z', l \leftarrow \mathbb{Z}_q$;
$E3^* : x' = \phi - (wy' + zz' + l(x_1 + wx_2 + \alpha(x_1 + wx_2 + y_1 + wy_2))) \mod q;^2$
$E6 : v \leftarrow \mathsf{HF}^\Gamma_{hk}(A, B, a, \hat{a}, c)$;
$E7' : d \leftarrow a^{x_1+y_1 v}\hat{a}^{x_2+y_2 v}$;
$\mathsf{return}(\psi = \langle A, B, a, \hat{a}, c, d\rangle, x', y', z', l)$.

$\mathsf{DecryptionOracle}(\psi := (A, B, a, \hat{a}, c, d))$
$D1$ : parse $\psi$ as a 5-tuple $(A, B, a, \hat{a}, c, d) \in Server \times \hat{G}^5$;
return reject and halt if it is not of this form;
$D2$ : if $A \in Server$ and if $B$, $a$, $\hat{a}$, $c \in G$;
otherwise return reject and halt;
$D3'$ : if $(A, B, a, \hat{a}, c) \neq (A^*, B^*, a^*, \hat{a}^*, c^*)$ and $v = v^*$,
where $v \leftarrow \mathsf{HF}^\Gamma_{hk}(A, B, a, \hat{a}, c)$ then $\mathsf{return}(A, B, a, \hat{a}, c)$ and halt,
otherwise return reject and halt.

$\mathsf{B}_{\mathsf{S1}}(\Gamma[\hat{G}, G, g, q])$
$A^* \xleftarrow{R} Server$;
$\phi \xleftarrow{R} \mathbb{Z}_q; B^* = g^\phi;^1$
$u \xleftarrow{R} \mathbb{Z}_q; a^* = g^u$;
$w \xleftarrow{R} \mathbb{Z}_q^*; \hat{g} = g^w$;
$\hat{u} \xleftarrow{R} \mathbb{Z}_q/\{u\}; \hat{a}^* = \hat{g}^{\hat{u}}$;
$r \xleftarrow{R} \mathbb{Z}_q; c^* = g^r$;
$\rho^* = (A^*, B^*, a^*, \hat{a}^*, c^*)$;
$\mathsf{return}(\rho^*, st := (\phi, u, w, \hat{u}, r))$.

$\mathsf{RealOrRandom}(m, S, \alpha)$
if $S \neq A^*$, $\mathsf{B}_{\mathsf{S2}}$ halt;
otherwise:
$\psi \leftarrow \mathsf{Encryption}(m, S, \alpha)$;
$\mathsf{return}(\psi)$.
```

[1] $B^* = g^\phi = g^{x'}g^{wy'}g^{zz'}.(g^{x_1+wx_2}(g^{y_1+wy_2})^\alpha)^l$

[2] $\phi = x' + wy' + zz' + l(x_1 + wx_2 + \alpha(x_1 + wx_2 + y_1 + wy_2))$

Figure 4.10: B_S algorithm regarding the server version Π_S.

advantage it has against a UOWHF:

$$Pr[\mathsf{B}win] \leq \varepsilon_{uow}(\lambda); \tag{4.15}$$

$$Pr[\mathsf{B}win|U_5]\frac{1}{|User|} \leq \varepsilon_{uow}(\lambda); \tag{4.16}$$

$$Pr[\mathsf{B}win|U_5] \leq |User|\varepsilon_{uow}(\lambda); \tag{4.17}$$

We know from [5] that $|Pr[R_5] - Pr[R_4]| \leq Pr[C_5]$. Furthermore, it is easy to see that the environment in which B runs the adversary conditioned on U_5, and up until C_5 occurs is identical to both games G_4 and G_5. In this extension $Pr[C_5] =$

by $\mathcal{A}$ doesn't match A^*.

$Pr[\mathsf{B}win|U_5] = Pr[\mathsf{B}win]$. Putting things together, we get

$$|Pr[R_5] - Pr[R_4]| \leq Pr[C_5] \leq |User|\varepsilon_{uow}(\lambda). \tag{4.18}$$

Finally, combining the previous result with the full security proof for the Cramer-Shoup cryptosystem presented in [5] we have reached an expression for the advantage of an adversary against the Extended Cramer-Shoup:

$$AdvCCA_{ExtCS,\mathcal{A}}(\Gamma) \leq AdvDDH_{\mathcal{G},\mathcal{A}_1}(\Gamma) + |User|\ AdvUOW_{\mathsf{HF},\mathcal{A}_2}(\Gamma) + \frac{Q_A + 3}{q}. \tag{4.19}$$

4.5 Global concrete security analysis

After the presented analysis of Lamport's One-time Signature, and the Extension to the Cramer-Shoup Encryption scheme, we recall the PAKE adversarial advantage described in equation 4.1. We will now perform a concrete security analysis around the PAKE protocol.

Extended Cramer-Shoup

We recall Extended Cramer-Shoup advantage in PAKE's context in equation 4.1, where we want to guarantee 128 bit security:

$$2q_{send}(AdvCCA_{ExtCS,\mathcal{A}}(\Gamma)) \leq 2^{-128}. \tag{4.20}$$

As previously defined, we will use $|User| = max(|Server|, |Client|) = 2^{10}$, $Q_A = 2^{40}$, $q_{send} = 2^{40}$. Plugging these values in to the advantage expression and combining equations 4.19 and 4.20, we obtain

$$\begin{aligned}
2q_{send}\left(\mathsf{AdvDDH}_{\mathcal{G},A_1}(\Gamma) + \varepsilon_{uow}(\lambda).|User| + \frac{2^{40}+3}{q}\right) &\leq 2^{-128};\\
\mathsf{AdvDDH}_{\mathcal{G},A_1}(\Gamma) + \varepsilon_{uow}(\lambda).|User| + \frac{2^{40}+3}{q} &\leq \frac{2^{-128}}{2q_{send}};\\
\mathsf{AdvDDH}_{\mathcal{G},A_1}(\Gamma) + \varepsilon_{uow}(\lambda).|User| + \frac{2^{40}+3}{q} &\leq \frac{2^{-128}}{2^{41}};\\
\mathsf{AdvDDH}_{\mathcal{G},A_1}(\Gamma) + \varepsilon_{uow}(\lambda).|User| &\leq 2^{-169} - \frac{2^{40}+3}{q};\\
\mathsf{AdvDDH}_{\mathcal{G},A_1}(\Gamma) + \varepsilon_{uow}(\lambda).|User| &\leq 2^{-169};
\end{aligned}$$

From these calculations, and assigning the same weight to each term on the left-hand side of the inequality, we reach:

$$\mathsf{AdvDDH}_{\mathcal{G},A_1}(\Gamma) \leq 2^{-170}; \qquad \varepsilon_{uow}(\lambda) \leq \frac{2^{-170}}{|User|} = 2^{-180}.$$

We need 2^{-170} security in $\mathsf{AdvDDH}_{\mathcal{G},A_1}(\Gamma)$ so, relying on the recommendations in [12], we stipulate a group size of roughy 9185 bits. In Extended Cramer-Shoup we need to assure the universal one-wayness of the hash function with a maximum advantage of roughly 2^{-180}. Following the recommendations provided by *ECRYPT II* [9] SHA-256 should be used.

Lamport's One-Time Signature

The concrete security analysis of Lamport's one-time signature is as follows. From equations 4.1 and 4.9 we know the advantage of $\mathcal{A}$ against the one-time signature, where we want to assure 128 bit security:

$$2q_{send}\left(Pr[SigForge^{\mathsf{sEUF-CMA}}_{\mathcal{A},\Pi}(\lambda) = 1]\right) \leq 2^{-128}.$$

The message length used in the one-time signature influences its security needs. Recalling figure 4.1, the signed message by the one-time signature is formed as $(\beta \mid K)$, where β is of hash output of size 256 bits, and $K \in \mathbb{Z}_q$, its length was 9185 bits, as calculated in the previous section. So the message length l will be $l = 9185 + 256 \approx 2^{14}$ bits.

$$\begin{aligned}
2q_{send}\ \varepsilon_{sig}(\lambda) &\leq 2^{-128};\\
2q_{send}Pr[SigForge^{\mathsf{sEUF-CMA}}_{\mathcal{A},\Pi}(\lambda) = 1] &\leq 2^{-128};\\
2q_{send}(2l(\lambda)\ \varepsilon_{tcr}(\lambda) + 2l(\lambda)\ \varepsilon_{owf}(\lambda)) &\leq 2^{-128};\\
2^1 2^{40} 2^2 2^{14}(\varepsilon_{tcr}(\lambda) + \varepsilon_{owf}(\lambda)) &\leq 2^{-128};
\end{aligned}$$

Considering SHA family of hash functions, we assume that $\varepsilon_{owf} \approx \varepsilon_{tcr}$. This allows to estimate ε as

$$\begin{aligned}
&\varepsilon(\lambda) \leq 2^{-128-40-14-2-1};\\
&\varepsilon(\lambda) \leq 2^{-185}.
\end{aligned}$$

Using a cryptographic hash function, in the absence of any analytic weaknesses, only brute force methods are available to the attacker. If n is the size of the hash outputs, one would from a secure hash expect around 2^n operations to be required to break the pre-image resistance and 2^{nd} pre-image resistance properties and around $2^{n/2}$ operations to break the property of collision resistance, due to the birthday paradox. Consequently, we need to choose a secure f with a large enough n so that breaking the second pre-image resistance property has a complexity over 2^{185}. Following the recommendations provided by *ECRYPT II* [9], SHA-256 should be chosen.

Putting things together

Finally we will calculate the advantage against the Decisional Diffie-Hellman in PAKE's (equation 4.1).

$$\begin{aligned}
2\varepsilon_{ddh}(\lambda) &\leq 2^{-128};\\
\varepsilon_{ddh}(\lambda) &\leq 2^{-128-1};\\
\varepsilon_{ddh}(\lambda) &\leq 2^{-129}.
\end{aligned}$$

For the Decisional Diffie-Hellman, we need 2^{-129} security. Using as reference [12], we need a group size of $q = 2^{3307}$. Furthermore, this would immediately give us very low values for the trailing terms in the definition of advantage, as can be seen below:

$$\begin{aligned}
\frac{min\{2q_{reveal}, q_{send}\}}{q} + \frac{2min\{q_{reveal}, q_{execute}\}}{q^2} &\leq 2^{-128};\\
\frac{2^{41}}{q} + \frac{2min\{q_{send}, q_{send}\}}{q^2} &\leq 2^{-128};\\
\frac{2^{41}}{q} + \frac{2^{41}}{q^2} &\leq 2^{-128};\\
\frac{2^{41}}{q} + \frac{2^{41}}{q^2} &\leq 2^{-128};\\
\frac{2^{41}}{2^{3307}} + \frac{2^{41}}{(2^{3307})^2} &\leq 2^{-128}.
\end{aligned}$$

Our conclusion is therefore that the group size we have defined earlier for the extended Cramer-Shoup scheme is more than enough.

4.6 Practical implementation

With the results achieved in the earlier sections, we have tested an implementation of the PAKE protocol[1] with the parameter sizes stipulated above. The results below were achieved on a machine with a two core 2.4 Ghz *Intel Core 2 duo* processor with 5GB of RAM. The source code was optimized with `gcc`'s `-O2` option. We achieved a very satisfiable time and memory consuming value, approximately 1 MByte of memory for each execution and the timings presented in table 4.1.

time	0m3.262s
	0m3.269s
	0m3.299s

Table 4.1: Execution times of PAKE's `C++` implementation.

[1] This implementation was mostly developed by Manuel Costa, prior to the beginning of this dissertation work.

Chapter 5

Conclusion

In this dissertation we have analyzed two key agreement protocols offering different security properties. These two protocols deal with weak sources of randomness in order to two parties to securely communicate in the presence of an adversary that controls the communication channel. In this thesis we assume that weak-secrets used in both protocols came from the same type of source, in this case from the common communication channel's noise.

The IT-AKA protocol, proposed by Dodis et al. in [7], is a key-agreement framework based on the notion of randomness extractors as its atomic function. The weak-secret is *mined* for randomness. Intuitively, each component uses a pre-determined number of bits from the weak-secret and for that bit length extracts some amount of randomness needed in a particular process of the protocol. For this methodology it is required that the weak-secret to have a minimum value of entropy to be able to assure that the key agreed has at least the minimum level of security expected. We have analyzed the various minimum length requirements in all of its building blocks, in order to consolidate the minimum length size for the weak source that would guarantee a global 128-bit security in the protocol. For each execution we need at least a 2^{16} bit weak secret, as the weak secret can only be used once. An efficient construction of extractor has not yet been achieved. So, to reach a practical implementation of the protocol we had to find a tangible substitute for the extractor. In our experiments we simply used a standard cryptographic construction based on SHA-256.

The PAKE protocol proposed in [11], has as main components an extension to the Cramer-Shoup protocol and a one-time signature. The security properties from the secret key created in PAKE are not derived directly by the randomness properties the weak-secret. Instead, this is used as a password. Key security is achieved by introducing fresh randomness in the protocol execution itself, and the weak-secret is used as a boot-strapper for the protocol. We made a concrete security analysis for the protocol's building blocks that gave us a concrete values to achieve a 128-bit security of global security. In particular, for the *one-time signature* we recommend a hash function with at least 185-bit output, so we recommended the use of SHA-256 hash function. In the case of the extension to the Cramer-Shoup cryptosystem, we needed group sizes of at least $q = 2^{9185}$ for the decisional Diffie-Hellman assumption to imply the intended level of security, and a hash function with a 180-bit length output, so it is recommended to use SHA-256 hash function.

Comparing the weak-secret size for both protocols, it is clear that the PAKE protocol needs a substantially smaller weak-secret. In the information theoretic setting, we have achieved values for the weak-secret size needed to derive a 128-bit secure key to be at least 2^{16} bits (8 kbytes) for each protocol execution, where each new communication would need a fresh weak-secret. In the case of PAKE protocol, it is only required that the weak secret retains sufficient entropy to ensure that a successful guessing attack is infeasible, and this can be adjusted according to the requirements of the application. Furthermore, a weak secret can be reused several times, without compromising its security.

Although we have constructed implementations of both protocols, we note that the two are not directly comparable, as they offer totally different security guarantees. The conclusion of our study is therefore that the typical tradeoffs between computational and information theoretic security in the encryption scenario for practical applications carry over to the WITS project setting when one considers the authenticated key agreement stage. An interesting direction for future work is to integrate the evaluated protocols with concrete implementations of physical-layer security mechanisms, in order to evaluate the feasibility of both solutions in a practical setting.

Bibliography

[1] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'00, pages 139–155, Berlin, Heidelberg, 2000. Springer-Verlag.

[2] Mihir Bellare and Phillip Rogaway. full version. collision-resistant hashing: Towards making uowhfs practical, 1997.

[3] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE SYMPOSIUM ON RESEARCH IN SECURITY AND PRIVACY*, pages 72–84, 1992.

[4] Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational diffie-hellman. In *In Proc. of PKC 2006*, pages 229–240. Springer-Verlag, 2006.

[5] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33:167–226, 2001.

[6] Yevgeniy Dodis, Jonathan Katz, and Leonid Reyzin. Robust fuzzy extractors and authenticated key agreement from close secrets. In *In Advances in Cryptology—CRYPTO '06*, pages 232–250. Springer, 2006.

[7] Yevgeniy Dodis and Daniel Wichs. Non-malleable extractors and symmetric key cryptography from weak secrets. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 601–610, New York, NY, USA, 2009. ACM.

[8] Venkatesan Guruswami and Christopher Umans. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. In *In Proceedings of the 22nd Annual IEEE Conference on Computational Complexity*, pages 96–108. IEEE Computer Society, 2007.

[9] ECRYPT II. Ecrypt ii yearly report on algorithms and keysizes 2008-2009, 2009.

[10] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[11] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 475–494, London, UK, 2001. Springer-Verlag.

[12] Arjen K. Lenstra. Key length, 2004.

[13] Ueli Maurer, Renato Renner, and Stefan Wolf. Unbreakable keys from random noise. In P. Tuyls, B. Skoric, and T. Kevenaar, editors, *Security with Noisy Data*, pages 21–44. Springer-Verlag, 2007.

[14] David Pointcheval. Computational security for cryptography, 2009.