BACHELOR THESIS
COMPUTING SCIENCE

RADBOUD UNIVERSITY

The X3DH Protocol: A Proof of Security

Author:
Ferran van der Have
s4104145

First supervisor/assessor:
Dr. B.J.M. Mennink
b.mennink@cs.ru.nl

Second supervisor:
Prof. Dr. J.J.C. Daemen
j.daemen@cs.ru.nl

Second assessor:
Dr. S. Samardjiska
simonas@cs.ru.nl

January 22, 2022

Abstract

The X3DH protocols is a key agreement protocol that is used in the Signal protocol to establish a shared secret key between two parties. While there is a rigorous security analysis for the Signal protocol, the X3DH protocol does not have the same level of analysis. In this paper we will provide such an analysis and provide explanations for most of the theory that is used. We will define the security this protocol gives and give a proof of security of the X3DH protocol by using game hops and introducing an adversary with power over the network and proving a bound on the advantage of the adversary. The conclusion is that the X3DH protocol provides secrecy of the shared secret and the message.

Contents

Chapter 1

Introduction

Messaging services are very popular and are used globally. However, privacy is becoming a bigger topic for the consumers of these services. In return, the providers of these services use cryptographic protocols to secure the messages and make it very difficult for adversaries to read these messages. This was not always the case, as the first messaging services were hardly encrypted at all.

One of the cryptographic protocols that is used nowadays is the Signal protocol. The Signal protocol is a secure messaging protocol that is included in many known messaging applications, such as Signal, Skype [27], WhatsApp [37] and Facebook Messenger [18]. This protocol is used to provide end-to-end encryption of communication, meaning that it aims to prevent third parties to read the messages while it is in transit from one party to another.

Initially, there was no proof of security, only a scheme when the protocol was implemented. Later, Cohn-Gordon et al. [11] were the first to provide a formal security analysis of the Signal protocol. While the protocol has been out for a number of years, there has only been a small amount of research done about the Signal protocol.

1.1 Signal Protocol

Signal consists of two protocols: an initial key agreement protocol Extended Triple Diffie-Hellman (X3DH) [29] and a *double ratcheting* algorithm [28].

1.1.1 Double Ratchet

The *double ratcheting* algorithm is used to encrypt messages based on a shared secret key, and consists of two stages: asymmetric and symmetric ratcheting. For every new message, a new key will be derived from the previous keys in such a way that previous keys cannot be calculated with the help of later keys.

In the asymmetric stage each party generates a secret value using the Diffie-Hellman protocol which is used to generate the key for the next message. The two

parties take turns in generating new secrets and sending them to the other party to use in deriving the shared secret keys.

In the case that a party wants to send multiple messages at once, the method used in the asymmetric stage cannot be used. For this case there is the symmetric stage. In this stage, every message will have a new key derived from the same secret.

The shared secret key which is used to start the *double ratcheting* algorithm is commonly received from the X3DH protocol. The X3DH protocol is used to agree on a shared secret key between two parties. Like other key agreement protocols, such as Diffie-Hellman or RSA-KEM, it is done in such a way that other parties that are listening in on the conversation will not be able to calculate this secret key. The focus of this paper is the X3DH protocol and this protocol will be further explained in Section 3.1.

1.1.2 Properties

Signal claims to have the following properties:

> Post-compromise security [12]: If at some point an adversary compromises a secret and is able to calculate the shared secret key that is used, then future communication is still secure. However this comes with a condition. If at some point an adversary compromises a secret key, but the adversary missed at least one message that was sent after the secret key has been compromised, then the adversary can no longer read the communication as a new secret has been introduced of which the adversary has no knowledge.
>
> Forward secrecy [9]: If at some point an adversary compromises a secret key, then past communication is still secure as the adversary would be unable to decode these past messages.
>
> Resilience (definition is taken from [4]): This means that the output of this protocol is computationally indistinguishable from random. In the case of a key exchange protocol, the output of this protocol is a shared secret key between two parties.
>
> Plausible deniability [10]: This means that there is no proof that a party has sent a message to another party, in other words two parties can deny that they have had communication.

Signal claims that the *double ratchet* algorithm provides post-compromise security, forward secrecy and resilience, and X3DH provides forward secrecy and plausible deniability.

1.2 Motivation

In the security analysis of Cohn-Gordon et al. [11], they use the Triple Dif e-Hellman protocol as key agreement protocol. It is also possible to use X3DH as key agreement protocol. The difference between the two protocols is that X3DH has an additional Dif e-Hellman value that is used to calculate the shared secret key.

In Triple Dif e-Hellman, the responder does not provide an ephemeral key or one-time prekeys. This makes replaying messages possible [29].

If the initiator sends an initial message without the use of a one-time prekey, then an adversary can replay this message multiple times. The responder will then think that the initiator send multiple messages, whereas the initiator only sent one.

While Cohn-Gordon et al. used the Triple Dif e-Hellman protocol as key agreement, they also proved secrecy for the X3DH protocol. The goal of this paper will be to focus on only on the X3DH protocol and prove the secrecy of this protocol.

1.3 Contributions

The aim of this thesis is to de ne the security that X3DH provides. It is unclear what security X3DH actually provides, so de ning this makes the protocol itself better to understand.

Another goal is to provide a clear and accessible proof of secrecy of the X3DH protocol, while also providing a clear de nition of secrecy in this protocol. The proof is based on the security analysis of Cohn-Gordon et al. [11]. This analysis is very extensive, but does not fully explain the underlying theory. The goal of the current project is to rst provide all the necessary knowledge that is needed and then provide the proof of secrecy for the X3DH protocol.

By making this proof more accessible, other cryptographers or researchers that use the X3DH protocol or research it will know what X3DH is and can verify the security it provides.

1.4 Related Work

This paper heavily relies on the research done by Cohn-Gordon et al. [11]. They provide a formal security analysis of the Signal protocol, and in this analysis they prove the security of the Triple Dif e-Hellman protocol, which is a protocol which is almost the same as X3DH, but misses one combination of secret values that X3DH has. They have given a proof for the security in the case that X3DH is used. Our goal is to provide a similar proof of security for the actual X3DH protocol and provide all the necessary preliminary knowledge for this proof.

In another work, Kobeissi et al. [23] use CryptoVerif and ProVerif to provide a framework to automate veri cation of cryptographic protocols. They used this

framework and applied it to a simpli ed version of the Signal protocol in order to nd weaknesses in this protocol.

Almuzaini et al. [2] provided another formal analysis of the Signal protocol using a model checking tool called Scyther. Whereas Cohn-Gordon et al. assumed that the secrets that are used are veri ed using another channel of communication, in this research it is not assumed so. Without this assumption Almuzaini et al. did nd an attack on the Signal protocol that exposes a secret key.

Our paper focuses on one-to-one chats using the Signal protocol, however it is possible to implement group chats, which Rösler et al. [34] focused on. They analyzed the group chat implementations of Signal and other messaging services which implement the Signal protocol.

Cremers et al. [13] have researched a clone attack on messaging services that implement the Signal protocol. A clone attack is an attack that clones the identity of a party after the full state of a party is compromised. The conclusion of this research is that many messaging services cannot recover from a clone attack, and as such will violate the post-compromise security claim.

The speci cations of the X3DH protocol by Marlinspike et al. [29] from Signal is used to de ne our model of X3DH and explain the X3DH protocol.

Chapter 2

Preliminaries

2.1 Mathematical structures

Cryptographic schemes are based on certain mathematical structures. In this section we will brie y explain the structures that we will use.

2.1.1 Groups

In this work, a group is a tuple $(A; /)$, where A is a set and $/$ a binary operation, with the following properties:

Closed:	$8x; y \; 2 \; A :$	$a / b \; 2 \; A,$
Associative:	$8x; y; z \; 2 \; A :$	$(a / b) / c \; = \; a / \; (b / c),$
Neutral element:	$9e \; 2 \; A; 8a \; 2 \; A :$	$a / e \; = \; e / a \; = \; a,$
Inverse element:	$8a \; 2 \; A; 9a^0 \; 2 \; A :$	$a / a^{\,0} = a^0 / a \; = \; e,$
Commutative:	$8a; b \; 2 \; A :$	$a / b \; = \; b / a .$

The commutative property is optional, however groups that satisfy this property are called abelian. A group can be in nite depending on A. The number of elements in A is called the order of the group. The order of an element $a \; 2 \; A$, denoted as $ord(a)$, is the smallest positive integer n such that $a^n \; = \; e$, or $[n]a = e$ in the case of addition. Modular addition is a group with the operation $+_n$ which is denoted as $(Z=nZ; +)$.

2.1.2 Cyclic Groups

Let $(A; +)$ be a group and $g \; 2 \; (A; +)$. Consider the set $f[0]g; [1]g; : : : ord(g) \quad 1g$. This set is a group, denoted hgi:

Closed:	We can see that $8i; j \; 2 \; f \; 0 : : : ord(g) \quad 1g$ it holds that $[i]g + [j]g = [i + j \mod ord(g)]g \; 2 \; hgi$.

Associative: Is trivial, since $+$ is associative.

Neutral Element: $[0]_g$ is the neutral element.

Inverse Element: We can see that $\forall i \in \{0 : : : \mathrm{ord}(g) \quad 1\}$ the inverse of $[i]_g$ is $[\mathrm{ord}(g) \quad i]_g$.

This group is called a cyclic group and g is called the generator of this group. For example, in a group $(Z=nZ; +)$, the generator $g = 1$. Cyclic groups are abelian, and a cyclic group of order q is isomorphic with $(Z=qZ; +)$. Isomorphic means that the groups have the same structure and properties. Having an isomorphism means that there is a mapping function from one group to the other which is a bijection.

A subset $H \subseteq G$ of a group $(G; +)$ is called a subgroup of G, if $(H; +)$ is also a group.

2.1.3 Multiplicative Groups

To have a group that is based on modular multiplication $(Z=nZ; \cdot)$ one has to remove some elements from $[0; : : : ; n \quad 1]$, as some elements do not have an inverse. The neutral element of a multiplicative group is 1. For example, 0 has no inverse. This means that if we have an element $i \in (Z=nZ; \cdot); i \neq 0$ and for some $j \in \mathbb{N}$ we have that $i \cdot j = n = 0 \mod n$, then i does not have an inverse either. The only numbers that have an inverse $(Z=nZ; \cdot)$ are numbers that are coprime to n. A number x is coprime to n if the greatest common divisor is 1. In the case that n is a prime, then all elements $\in \{1 : : : n \quad 1\}$ are coprime to n, so we only have to remove 0. We will define such modular multiplication groups with the elements that violate the group properties removed as $(Z=nZ)^{\cdot}$.

2.1.4 Fields

In this work, a field is a triple $(F; +; \cdot)$, where $(F; +)$ is a group and $(F \setminus \{0\})$ is a group and distributivity holds. Distributivity means that for $a; b; c \in F$ it holds that $a \cdot (b + c) = a \cdot b + a \cdot c$. The field $(Z=pZ; +; \cdot)$, where p is a prime, is called a prime field and is denoted as $GF(p)$. There is a unique prime field for each prime number.

2.1.5 Elliptic Curves

In public-key cryptography, parties have a public key and a private key. The first is publicly known and the latter is a secret. Some public-key cryptography is based on the fact that solving discrete logarithm, explained in Section 2.3.1, is computationally difficult in certain cyclic groups. When public-key cryptography was introduced, multiplicative groups based on prime numbers were used.

Another form of public-key cryptography is Elliptic Curve Cryptography (ECC) [14], which is based on subgroups of elliptic curves. Elliptic curves [22] make it possible to use smaller keys compared to non-EC cryptography, while still being equally

hard to break. Breaking a cryptographic protocol means that an adversary can calculate or deduce the shared secret, without having access to the secret components that make up this shared secret.

Let $GF(q)$ be a finite field with q elements, where $q = p^m$, p is prime and $p \neq 2, 3$. An elliptic curve over $GF(q)$ is the set of pairs (x, y) that satisfies the following equation, called the Weierstrass equation:

$$y^2 = x^3 + ax + b. \tag{2.1}$$

Figure 2.1: Addition in elliptic curves: $P + Q = R$

Figure 2.2: Doubling in elliptic curves: $2P = R$

Elliptic Curve Group

We can now define two operations. First we have addition of two points on an elliptic curve, which results in a third point on the curve. It is best explained by a geometric example instead of formulas shown in Figure 2.1. We have points P and Q. To add these points together we draw a straight line through both points. This line will intersect on a third point on the curve, on point R'. Now we take the reflection of R' on the x-axis to get R. This R is the result of the addition of P and Q.

The second operation is doubling, which is also better explained by a geometric example, shown in Figure 2.2. We have a point P which we are going to double. First we draw the tangent line of the elliptic curve at point P, which intersects at point R'. Next we take the reflection of R' on the x-axis to get R, which is the result of the doubling of P.

Now we can define an abelian group $(E; +)$ of an elliptic curve defined over a field, with a neutral element that is a called point at infinity, 1. This point is the sum of P and $-P$ (which is the reflection of point P on the x-axis) and is a point on every vertical line. We must show that the group properties hold:

Closed:	By construction adding two points results in another point on the curve E.
Associative:	This is not easy to prove. One of the proofs is given by Friedl [21].
Neutral Element:	$\forall P \in E$ it holds that $P + 1 = 1 + P = P$.
Inverse Element:	$\forall P \in E$ it holds that $P + (-P) = 1$.
Commutative:	This is trivial. For example, in Figure 2.1 it does not matter if we draw the line through P first or Q.

An elliptic curve over a field can be defined by so-called domain parameters. These domain parameters are a sextuple $T = (p; a; b; G; n; h)$, where the elements are as follows:

p specifies the finite field $GF(p)$.

a and b specify the curve E.

$G \in E - GF(p)$, which is a point on the curve over $GF(p)$, is a base point. This can be any point, but must be agreed upon by both parties.

$n = ord(G)$.

$h = \#E - GF(p) = n$. This is called the cofactor of the elliptic curve E.

X3DH is implemented with either the Elliptic Curve Diffie-Hellman X25519 or X448 [1]. These curves use the Montgomery form representation of elliptic curves.

$$By^2 = x^3 + Ax^2 + x: \tag{2.2}$$

Both use a function "X 25519" and "X 448", respectively, in order to do scalar multiplications on the curve. They take as an input a coordinate of a point and a scalar n and outputs nP. In order for Alice to generate a public key in X25519, first she generates a private key a and converts it into a byte string of 32 bytes. Then her public key is $A = aG$, where G is the base point, which is the point with the minimal, positive x value in the case of these functions.

Then her public key is $A = X25519(a, 9)$ where 9 is the x-coordinate of the base point. Bob does the same with a private key b and computes B. Now both can compute the shared secret key $= X25519(a, B) = X25519(b, A)$.

2.2 Key Derivation Functions

In order to get a shared secret key, a Key Derivation Function (KDF) is often used to derive multiple shared secret keys from a shared secret. A KDF can stretch shared secrets into longer strings and can create multiple different secret keys from the same shared secrets with the help of a salt.

This function uses as input a secret value and possibly known parameters. In this work, a KDF accepts an input of four arguments: a value from keying material, a length value ℓ, an optional salt value and an optional context variable. The last two optional arguments can be set to a constant if they are not needed. The salt value is a non-secret random value that provides additional randomness to the function. Adding a salt value makes the KDF stronger, by making it less predictable, as using a salt makes different uses of the KDF independent of each other, provided that the KDF is strong enough. With a salt we can even derive two different outputs from the same secret value input, which is called domain seperation. The context value can provide information about the context the KDF is used in. For example, it could contain identities of the user, which algorithm is used or even a timestamp.

The output of a KDF is a string of bits with length ℓ which can be used in further protocols as a key. A KDF is a one-way function, meaning that if we know the output of a KDF, it is infeasible to calculate the input values that were used. X3DH also makes use of a Key Derivation Function (KDF), namely the HKDF algorithm [25, 26].

2.3 Computational Problems

Most cryptographic schemes are based on computationally hard problems. These computationally hard problems cannot be solved efficiently, which means that if a cryptographic scheme is based on such a problem in a successful way, then breaking such a scheme could not be done efficiently either. This is done by design, because having a cryptographic scheme that takes a long time to break is desirable.

2.3.1 Discrete Logarithm

Let G be a cyclic group with generator g and order q. Then we know that for every $y \in G$, there is a unique $x \in (\mathbb{Z}/q\mathbb{Z})$ such that $g^x = y$. This value x is called the discrete logarithm of y with respect to g: $x = \log_g y$. Note that x is unique in $(\mathbb{Z}/q\mathbb{Z})$, which means that if there is x' where $g^{x'} = y$, then $x' \equiv x \bmod q$.

The discrete logarithm problem [30] in a cyclic group G with generator g and order q is as follows: given a randomly chosen $y \in G$, compute $\log_g y$. In other

words, find $x \in (\mathbb{Z}/q\mathbb{Z})$ such that $g^x = y$.

In the case of Elliptic Curves [36] it is as follows: consider an elliptic curve E over a finite field $GF(p)$ with p a prime and $P = (x, y)$ a point of prime order n, meaning that n is the smallest number such that $nP = 1$, on the elliptic curve. Let k be an integer. Let E be the cyclic group generated by P. Given a point $Q = kP \in E$, it will be hard to compute k.

2.4 Random Oracles

In order to analyse the security of cryptographic protocols, a random oracle is often used. Random oracles were introduced and first used in proofs by Bellare and Rogaway [6]. These oracles are a black box, which means we know nothing of their inner workings. In the definition of Bellare and Rogaway, the random oracles produce a bit-string of infinite length: $RO : \{0,1\}^* \to \{0,1\}^\infty$, of which we then take the first n bits. In our work however, we focus on random oracles with fixed-length output: $RO : \{0,1\}^* \to \{0,1\}^n$.

Interacting with this oracle is done by queries. A party can query a string x to the random oracle and the oracle will see if x has been queried before. If it has not been queried before, the oracle will reply with a random string y of length n, where $n \in \mathbb{N}$. If x has been queried before, then the oracle will reply the same string that was returned when x was queried for the first time.

2.5 Distinguishability

With the help of random oracles we can conduct an experiment to help us prove the security of protocols. Formally in these experiments, the adversary Eve is an algorithm A. To measure the capability of the adversary, we use the following experiment.

Secretly we will select either our cryptographic protocol CP with a secret key or a random oracle RO. We choose so by generating a random bit b which will either be 0 or 1. We select CP if $b = 1$ and select RO if $b = 0$. It is up to the adversary A to find what the value of the bit is.

The experiment is shown in Figure 2.3. Now A will be in one of two worlds: the “real world” or the “ideal world”. In the real world, A interacts with CP and A interacts with RO in the ideal world. A does not know in which world it is in, but can query the entity it is interacting with, noted as “oracle”. When A queries input X, CP will process the input paired with the secret key which is unknown to the adversary and provide output Y of length n bits. RO will take input X and see if it has already received X before. If that is the case, output Y will be the same as when X was queried before. If X was not queried before, output Y will be a random bit string of length n. Eventually A returns a bit $b' \in \{0,1\}$. $b' = 1$ if A thinks it is in the real world, and $b' = 0$ if it thinks it is in the ideal world.

Figure 2.3: Distinguishability experiment with input X, output Y and adversary A.

In the case of public key cryptography, we can use a random oracle to prove the security of our protocol. If we give this random oracle a proper interface so that it accepts the same values as our protocol and provides similar looking output, we can query the random oracle for a session key. We can then use the experiment defined above to get a session key from either the protocol or the oracle and ask an adversary to tell us in which world he is in. Using this distinguishability experiment we then prove the security of the "ideal" version where we use the oracle. Then we will prove our "real" version secure, where we use our protocol, as otherwise we will have a efficient solution to a computational hard problem.

2.5.1 Advantages

If the adversary would just guess, then he has a success probability of $1/2$. So in order to have a successful attack, he needs to guess b with a probability that is significantly larger than $1/2$.

One can define the advantage of the adversary with this equation:

$$Adv_{CP}(A) = 2Pr(b = b') - 1. \tag{2.3}$$

However, in order to better work with the advantage, another formula is derived as follows. In this derivation we use the logical "and" symbol $\wedge$. This means that in order for the statement $a \wedge b$ to be true, both a and b must be true. We can derive

the following formula:

$$\begin{aligned}
\mathrm{Adv}_{CP}(A) &= 2\Pr(b = b') - 1 \\
&= 2\left[\Pr[b' = 1 \wedge b = 1] + \Pr[b' = 0 \wedge b = 0]\right] - 1 \\
&= \frac{\Pr(b' = 1 \wedge b = 1)}{\Pr(b = 1)} + \frac{\Pr(b' = 0 \wedge b = 0)}{\Pr(b = 0)} - 1 \qquad (2.4) \\
&= \Pr(b' = 1 \mid b = 1) + \Pr(b' = 0 \mid b = 0) - 1 \\
&= \Pr(b' = 1 \mid b = 1) - 1 + \Pr[b' = 0 \mid b = 0] \\
&= \Pr(b' = 1 \mid b = 1) - \Pr(b' = 1 \mid b = 0).
\end{aligned}$$

This formula is most used in practice in order to calculate the advantage of an adversary. In case of just guessing, $\mathrm{Adv}_{CP} = 0$. So if there is an attack where Adv_{CP} is significantly larger than 0, then this would count as a successful attack.

2.5.2 Complexity

The advantage of A of distinguishing between CP and RO increases when A makes queries to its oracle, called online queries. All these queries are stored in a query history, noted QH_d. To express the advantage we use the total number of bits sent to and received from the oracle for all queries, $|QH_d|$. This is called the data complexity.

It is also possible for A to implement CP itself with an arbitrarily chosen secret key. A does not have to be connected to its oracle for this, so these are called offline evaluations. These evaluations are stored in another query history, noted QH_c. The advantage in this case depends on the length of the key, $|QH_c|$. This is called the computational complexity.

Implementing and running the individual evaluations often involves more computations related to the scheme that is used. However the time these take is often neglected.

2.5.3 Security Strength

Proving an upper bound for the advantage of an adversary against the protocol is not possible. We can never know if a protocol is secure, we can only disprove the security of a protocol by providing an attack. The best we can do is claiming an upper bound that corresponds to an exhaustive key search. An exhaustive key search is where an adversary tries all possible keys until the right key is found. This is by no means efficient, but it will always succeed given enough time.

In the case we have a protocol P with a key length of k the upper bound we can claim the following:

$$\mathrm{Adv}_P(A) \leq \frac{|QH_c|}{2^k}. \qquad (2.5)$$

The adversary that matches this bound, is the adversary that performs an exhaustive key search.

2.6 Dif e-Hellman

In order to understand the Extended Triple Dif e-Hellman protocol, or X3DH, we need to understand the Dif e-Hellman protocol. Dif e-Hellman [19] is a key-exchange protocol, created by Whit eld Dif e and Martin Hellman dating back to 1976 [17]. The protocol ensures that two parties can calculate a shared secret, from their own personal secrets. A third party with no access to these personal secrets is unable to calculate the shared secret. Traditionally the two parties are named Alice and Bob and the malicious third party is named Eve.

Dif e-Hellman makes use of public key cryptography in order to create a shared secret that only Alice and Bob know. In order for Alice to send a message to Bob, she calculates the shared secret and uses this as an input for a Key Derivation Function (KDF). This is a function that creates a secret key based on the shared secret that can be used for secure communication. KDF is explained in Section 2.2. Bob can also compute the shared secret and uses the KDF to calculate the same secret key.

The protocol makes use of either a multiplicative group G of order q and a generator g or an elliptic curve group of order q and base point g. The protocol goes as follows:

Key generation: Alice chooses a number a and calculates $A = g^a \bmod q$.
Bob chooses a number b and calculates $B = g^b \bmod q$.

Alice sends A to Bob.

Bob sends B to Alice.

Now both parties can calculate the shared secret: $K = A^b = g^{ab} = g^{ba} = B^a \bmod q$.

The domain parameters of the Dif e-Hellman protocol are (q, g). These parameters de ne the group that is used in the protocol. The public keys are (A, B) and the private keys are (a, b).

We will use the notation $DH(A, B)$ for the the shared secret of both parties with the public keys A and B. It is important for each party to know the corresponding private key for one of the public keys.

2.6.1 Security

The Dif e-Hellman protocol is based on the computationally hard Dif e-Hellman assumption. This means that an attacker needs to solve this computationally hard problem if it wants to break the protocol. However this is dependent on the choice of the group and the generator. If the group is too small, solving the computational problem is not dif cult.

In order for Eve to nd out what the shared secret is, she has to either compromise one of the private keys, or nd a way to calculate the shared secret with only

the public keys. For example, in order for Eve to compromise the shared secret key g^{ab}, she has to either calculate a from g^a with a discrete logarithm, or Eve can have another method to calculate g^{ab} from g^a and g^b.

This means that breaking the Dif e-Hellman protocol by trying to compute the shared secret key is not ef cient and as such Dif e-Hellman is secure against these kinds of attacks.

2.6.2 Static and Ephemeral

There are multiple versions of the Dif e-Hellman protocol used in different scenarios. There are two static versions, namely static-static and ephemeral-static [32]. Static means that the private and public key are unchanged by the party for all communications. In the static-static case that means that the key generation is not part of the protocol, but must be done beforehand. In ephemeral-static, one party has a static key pair, while the other party creates new key pairs for every time a DH step is done in the protocol that is used.

There is also a ephemeral-ephemeral version, where each party generates a new key pair for each DH step done in the protocol. In the ephemeral versions the key generation is part of the protocol. These ephemeral keys can still be used for a longer time. For example in TLS [33], the ephemeral keys are used in the handshake protocol and then the keys that are derived from the handshake are used for longer sessions.

2.6.3 Public Key Authentication

It is possible, however, for Eve to impersonate Alice, by just sending Bob a public key and saying she is Alice. Therefore, it is important for Bob to authenticate the public key that was sent to him. One way of doing this, is by using an out-of-band authentication. This is a veri cation method that is on a different communication channel, for example Bob can physically meet Alice and compare the public key he received with the public key of Alice herself. This is dif cult to do on a large scale, especially if keys are replaced often or the parties that communicate with each other live very far away. Meeting thousands of people physically to check the public keys is very dif cult to do, especially in times of the Corona virus.

Signatures

A signature scheme [20] is a way to cryptographically sign a message in such a way that the receiver can authenticate that the message came from the sender and that the message is not changed. The use of signatures provide authentication of the party that is sending the message. It also provides integrity of the message, meaning that it is visible if the message is altered during transmission. It also provides non-repudiation of origin, meaning that the sender cannot deny having sent the message.

The problem with using signatures for authentication is that the public key that is used has to be authenticated itself before it can be used.

Signature schemes make use of the public and private keys of public key cryptography. If Alice sends a message to Bob, but wants to sign it, she signs the message with Alice her private key and sends both the signature and the original message to Bob. Bob can verify the signature using Alice her public key. Only Alice has access to her private key, so Bob can be almost sure that the message came from her.

Certificates

Another way of authenticating is by introducing a trusted third party (TTP), for example a company, also called a Certificate Authority (CA), and certificates. A certificate is a signature that binds an identity to a public key. For example, Alice and Bob have generated key pairs (a, A) and (b, B). If Alice is sure that Bob his public key is B, she can sign a message containing the public key of Bob and Bob his identity. For example, the message could be "Bob's public key" is B. Bob can then use the signature from Alice to show other parties that he is communicating with that B is indeed his public key. However, for this to work Alice needs to be trusted by the other parties that Bob is communicating with.

A CA can be such a party that is trusted by every other party. Any party that wants to use a CA to obtain and verify certificates must obtain the public key of the CA. This has to be done in a secure way, else it is not possible to authenticate the public keys of other parties using this CA. This can be done by physical means and is better to scale compared to the previous method, since every party only has to visit the CA once and they do not have to physically meet the other parties.

With the help of a CA, two parties Alice and Bob can authenticate the public keys of each other with the help of the certificates. Since Alice trusts the CA, she trusts the certificate that says that the public key of Bob B is

The way the Signal protocol handles certificates is dependent on the implementation. However, the common way for mobile applications, and as such messaging services implementing the Signal protocol, is to implement a CA inside the application.

2.6.4 Diffie-Hellman Problems

In the case of Diffie-Hellman an attacker wants to calculate the shared secret $K = g^{ab}$, where a and b are the private keys of the communicating parties. There are several variants of computational problems [3] that arise for the attacker. We will discuss three of these problems.

First there is the **C**omputational Diffie-Hellman Problem (CDH): given a triple of elements $(g; g^a; g^b)$ where a and b are randomly chosen, find $K = g^{ab}$. This is based on the discrete logarithm problem. If an adversary could easily solve the discrete logarithm problem, then the adversary can also solve CDH easily. First the

adversary uses the discrete logarithm to calculate a from g^a. Then the adversary can compute $(g^b)^a = g^{ab}$. However, there is no proof as of yet that this is the only way to solve CDH. In some special cases it can be shown that the discrete logarithm assumption is the same as the CDH assumption [15].

Second there is the Decisional Diffie-Hellman Problem (DDH) [8]: given a quadruple of elements (g, g^a, g^b, g^c) where a and b are randomly chosen, decide whether $c \equiv ab \bmod p$. In other words, if we have (g^a, g^b, x) it is difficult to decide if $x = g^{ab}$ or x is a random element in the group generated by g. It is the case that if CDH is easy to solve, then so is DDH, however the opposite is not true. There are some groups where solving DDH is easy, but CDH is not. These groups are called Gap Diffie-Hellman groups.

There is also a computational problem related to these Gap Diffie-Hellman groups, namely the Gap Diffie-Hellman Problem (GDH) [31]: given a triple of elements (g, g^a, g^b) where a and b are randomly chosen, find $X = g^{ab}$ with help of a DDH Oracle. Having access to a DDH Oracle means that we can query the oracle with a quadruple of elements (g, g^a, g^b, g^c). The oracle will return 1 if $g^c = g^{ab}$ and 0 otherwise. This can be used to verify potential solutions to the GDH problem and query them to the DDH oracle to see if they are correct. The proof of security of X3DH makes use of the GDH hardness assumption.

2.7 Models

2.7.1 Threat Models

A threat model is a model which contains the capabilities of the adversary. We define the capabilities and we will prove in the end that an adversary with these capabilities cannot break the protocol which you are trying to prove to be secure. For example, an adversary can be a passive attacker, meaning that the adversary can only eavesdrop. An adversary can also be an active attacker, meaning that the adversary can delete, modify, replay messages and has full control over the network.

2.7.2 Security Models

A security model is a model that can be used to model the protocol you are trying to prove secure. This contains variables and definitions of the protocol in such a way that you can prove security of the model.

2.8 Game Hopping

In this proof we will calculate the advantage of the adversary using game hopping. Game hopping is a technique used in security proofs of protocols [7, 35]. We will construct a series of games in order to prove the security. Each game differs slightly from the preceding game. These changes have to be small in order to analyze them

better. The advantage that the adversary has in each game is bounded in some way by the preceding game. The first game will be the original experiment. The following games will lead to a game for which we can prove a bound. Because each game is bounded in some way by their preceding games, we can then prove a bound for the original experiment.

2.8.1 Large Failure Events

Some of the game hops used in the proof are based on *large failure events* [16]. In this case we assume that two games will appear identical until a certain error event occurs, but the probability that this event occurs is very large. We also assume that this error event and winning the game are independent of each other. We can then calculate the advantage as follows: Let E be the error event that can occur in Game 1, and S_i be the event that you succeed in game i. Let Game 1 and 2 be identical until E occurs, and if E occurs then the adversary loses. If E does not occur, then the adversary wins in Game 2 if and only if it would win in Game 1. We can then see:

$$\begin{aligned} \Pr(S_2) &= \Pr(S_1 \wedge \neg E) \\ &= \Pr(S_1) \cdot \Pr(\neg E). \end{aligned} \tag{2.6}$$

This can be rewritten as:

$$\Pr(S_2) \cdot \frac{1}{\Pr(\neg E)} = \Pr(S_1). \tag{2.7}$$

However, this means that by rewriting the formula we must have $\Pr(\neg E) \neq 0$. It is the case that if $\Pr(\neg E) = 0$, then we see that $\Pr(S_2) = 0$ in equation 2.6.

For example, in Game 1 the attacker must calculate the shared secret key with a public key. In Game 2 the challenger guesses a public key $i \in \{1, \ldots, q\}$ and the attacker can only win if that public key is used. Then we can define the event E as the event that that certain public key is not used. As the probability that E occurs is $\frac{1}{q}$, we obtain that:

$$\Pr(S_1) = \Pr(S_2) \cdot q. \tag{2.8}$$

2.8.2 Small Failure Events

Other game hops are based on *small failure events*. As in large failure events, two games will be identical until a certain error event occurs. However, in this case the probability that this event occurs is small. In the case of small failure events we can use the following lemma, provided by Shoup [35]:

Lemma 1. Let A, B, F be events defined in some probability distribution, and suppose that $A \wedge \neg F \Leftrightarrow B \wedge \neg F$. Then $|\Pr(A) - \Pr(B)| \leq \Pr(F)$.

Chapter 3

Research

3.1 The X3DH Protocol

Extended Triple Dif e-Hellman or X3DH is a key agreement protocol. This means that through this protocol two parties can agree on a shared secret key through an unprotected channel of communication. In order to make the explanations easier to read, we will give names to the parties that are involved in the protocol. In cryptography it is common to name these parties Alice and Bob. X3DH is designed for asynchronous messaging. It is possible to use Dif e-Hellman for asynchronous messaging, however it is not designed speci cally for this. Asynchronous messaging makes it possible for Bob to have published some information on a server and be of ine, while Alice can establish a shared secret key and send encrypted messages using the information on the server. Figure 3.1 provides a visual representation of the X3DH protocol.

The server needs to be a trusted server in order to ensure secrecy and a secure connection. If that is not the case, then the server could provide a key to Alice of which the server knows the private key. In that case, the server would be able to read the messages.

3.1.1 Trusted Server

When we speak of trust in cryptography, we assume that the entity that is trusted will behave according to a set of rules. This set of rules is dependent on the context in which the entity acts. So in our case a trusted server will distribute the keys in the way that we expect and will not provide keys so that the server can read the messages.

If the server is not a trusted server, then the messages that are sent between Alice and Bob could be read by the server, or the messages can be deleted by the server.

In order to establish trust between a party and a server, it is possible to use the Trust On First Use (TOFU) principle. In short, a party connects to a server which the party does not trust yet and hopes nothing goes wrong. If nothing goes wrong

Figure 3.1: A visual representation of the X3DH protocol. The steps shown in the figure do not have to be done at once. In step 1 the parties publish their information to the server. In step 2 Alice requests Bob his information. In this step Bob can be offline. In step 3 Alice sends Bob the secrets needed to compute the shared secret. In this step Bob can also be offline and will receive this message from Alice when he comes online.

at first, then the server can be trusted. This principle is not the best principle to use, but it is the easiest one to use.

In this paper we do not assume that the server is trusted. However we do assume that Alice and Bob have authenticated the public keys of each other, so that an impersonification of either party by the server would be noticed.

3.1.2 Prekeys

We assume that Alice is the party that initiates the X3DH protocol and wants to send a message to Bob. In order for Bob to be able to receive a message, he needs to generate and store some keys on the server. The generation of these keys is implementation specific, but as specified in the documentation of Signal [29], they must either all be in X25519 form or in X448 form. These forms have been discussed in Section 2.1.5. Bob has a private key for every key that is uploaded to the server. First, Bob needs to generate and store his public identity key IK_B to the server. The private key corresponding to the identity key is ik_B and Bob should keep this key a secret. The identity key is a long-term key that Bob generates when he connects to the server for the first time. This key is tied to his identity so this key will not be replaced over time.

The second key that needs to be generated and stored is a medium-term public key SPK_B, called a signed prekey. This is called a signed prekey, because Bob also needs to store a public key certificate of SPK_B using IK_B. This signed prekey and the certificate have to be replaced regularly, by generating a new signed prekey. The server will be updated to use the new signed prekey. After replacing the key, Bob keeps the old private key of SPK_B for a short time, in order to be able to decrypt delayed messages. It is important that he deletes the old keys after that short period of time, in order to provide forward secrecy as these keys are needed to recalculate the shared secret that is used to decrypt past messages. The amount of time that the old keys are stored are dependent on the implementation. Since one-time prekeys are used, replaying older signed prekeys are not harmful, since they have to be accompanied with a one-time prekey. If Bob notices a one-time prekeys has been reused, Bob does not accept the message.

These one-time prekeys of OPK_B are the last keys that have to be uploaded by Bob. These keys are used a single time, after which the server deletes the key. When the server notices that the amount of one-time prekeys is getting low, or whenever Bob wants, he can upload more of these keys to the server. These keys and SPK_B are called "prekeys" because the keys are uploaded prior to the beginning of the protocol. So in short, the values Bob needs to store are:

- IK_B,
- SPK_B,
- a certificate of SPK_B using ik_B, the private key corresponding to IK_B,
- multiple OPK_B's.

The first two keys and the certificate combined with a single OPK_B is called a prekey bundle.

3.1.3 Initiating the Protocol

In order for Alice to initiate the protocol and send a message to Bob, she first has to request a prekey bundle. Alice needs to verify the certificate before she will use this bundle. Since she has access to IK_B, she can check the contents of the certificate. If the contents are equal to SPK_B, then the certificate is correct. If the certificate is incorrect, then Alice will not accept the message.

After checking the certificate, Alice creates an ephemeral Diffie-Hellman key pair with public key EK_A. Ephemeral keys are newly generated keys for each new instance of the protocol and are only used for that instance. Identity keys and medium-term keys can be used for multiple instances of the protocol, but ephemeral keys are only used for a single instance. Alice also has her own identity key IK_A of which she has the private key. Alice has access to the following private keys: ik_a and ek_a. Now she can compute the Diffie-Hellman secret keys that are needed to calculate the shared secret key sk_{AB}.

$$K_1 = DH(IK_A; SPK_B),$$

$$K_2 = DH(EK_A; IK_B),$$

$$K_3 = DH(EK_A; SPK_B),$$

$$K_4 = DH(EK_A; OPK_B).$$

See Figure 3.2 for a visual representation.

Figure 3.2: The first and last row are public keys that are from Bob and Alice respectively. The lines from the Diffe-Hellman keys indicate what keys are used in the calculation.

Using a KDF Alice and Bob can subsequently calculate the shared secret key: $K_{AB} = KDF(K_1 || K_2 || K_3 || K_4; `; salt; context)$, where $||$ means concatenation of the keys. Without loss of generality we assume that injective padding is used on the input. The $`$, salt and context variables are all dependent on how this protocol is implemented. After Alice has calculated K_{AB} she deletes her ephemeral key, ek_A, and all the K_i.

Now that Alice has the shared secret key, K_{AB}, she can send a message to Bob so that Bob can calculate K_{AB} as well. Alice will send the following to Bob:

IK_A,

EK_A,

an identifier of which one-time prekey is used,

an initial ciphertext that is encrypted with K_{AB}.

After the initial message, K_{AB} is used in the ratcheting algorithm to derive the future keys. Alice can append optional information, such as Bob his username or certificates. This is however dependent on the implementation.

3.1.4 Deciphering

In order for Bob to decipher the message, he needs to calculate K_{AB}. After receiving the initial message, Bob knows IK_A, EK_A and which one-time prekey Alice used. He also has access to the following private keys: spk_B, ik_B and opk_B. Now Bob can do the same Diffie-Hellman calculations that Alice did, because the public keys that are used are either already known by Bob or sent by Alice. It is important that Bob also deletes the intermediary values and opk_B after calculating K_{AB}. Now Bob can decrypt the initial ciphertext sent by Alice and checks if it is correct. It is important that if it is incorrect, for example if the message decrypts to a random string, Bob stops all communication and deletes all associated values. If it is correct, both parties can use shared secret key K_{AB} for the rest of their communication.

3.2 Defining the X3DH Protocol

Before we can consider the proof, we have to understand the context and what it is exactly we want to prove. We do so by providing a threat model in Section 3.2.1 which analyses possible adversaries and what it is we want to prove. Then we provide a security model in Section 3.2.2 which we will apply to model the X3DH protocol. A brief overview of the X3DH protocol can be seen in Figure 3.1. The models we provide are based on the models Cohn-Gordon et al. [11], however they analysed the entire Signal protocol, whereas we will only analyse the X3DH protocol.

3.2.1 Threat Model

We first need to define our threat model. We assume an active adversary Eve that has control over the network on which the messages will be sent. This means that Eve can delete messages, but also alter them. Eve can set up a session between two parties and can choose ephemeral keys. This way, if Eve cannot break the protocol, then attackers with less control over the network cannot either. We do not consider the use of side-channel attacks, for example a timing attack [24]. These attacks are focused not on the protocol itself, but on the devices that run the protocol. Our threat model is the same model that is used in the proof by Cohn-Gordon et al.

We assume that the KDF function is working as it should and that there is no other way to calculate the shared secret key than by using the Dif e-Hellman values.

We do assume that Alice and Bob have both veri ed the public keys of each other.

The property we want to prove is secrecy of the shared secret key. Authentication will be proven alongside secrecy by showing that only the intended parties could compute the key SK_{AB}. Secrecy of the shared secret key will be proven by showing that the shared key will stay a secret even if some secret values from either Alice or Bob would be compromised.

3.2.2 Security Model

In our proof we will use a *key indistinguishability* model [5]. Key indistinguishability means that it should be very hard for an adversary to distinguish between a real key and a random string of the same length. In our model we do not consider the certi cate that is used for authentication, as this certi cate is used for authentication and we assume that both parties have veri ed the public keys. Our security model is based on the model that is used in the proof by Cohn-Gordon et al. The difference is that Cohn-Gorden et al. modelled the entirety of the Signal protocol, whereas we only model the X3DH protocol. As such the de nitions are similar to the ones used in the paper of Cohn-Gordon et al. but are not entirely the same. One big difference is that we only have to model one stage, where in order to model the Signal protocol you have to model multiple stages.

There are three stages in the protocol. First the parties upload their prekey bundles, containing their identity key, signed prekey, a public key certi cate of the signed prekey and multiple one-time prekeys. The second stage is Alice requesting Bob's prekey bundle from the server. The third stage is Alice sending the message to Bob containing the keys needed to calculate a shared secret key.

Now we can de ne the protocol in terms of algorithms.

De nition 1. A key exchange protocol is a combination of algorithms with a set P containing all possible keys and a value e which states how many random bits are required in a session.

In our setting, the protocol consists of two key generation algorithms and two algorithms to run the protocol:

IdentityKeyGen(), which outputs the identity key pair (IK, ik),

PreKeyGen(ik), which takes ik as input and outputs the signed prekey pair (SPK, spk),

Initiate$_{ephemeral}(ik, spk, role, targetid)$, where ik and spk are the private keys of the identity key and the signed prekey, $role$ indicates whether this party is initiating a conversation or is responding to it. In the case of the

initiating party, there is also an identification of who the targeted responder is. The *ephemeral* is the ephemeral private key that is used. With the input and this *ephemeral* the algorithm creates the output, which is a state and possibly a message. The goal of the Initiate algorithm is to define the first stage of exchanging the prekey bundles.

$Run_{ephemeral}(ik, spk, state, message)$, where ik and spk are the private keys of the identity key and the signed prekey, *state* is the state of the protocol, and *message* is an incoming message. This algorithm also uses the ephemeral private key to create the output. The output is an updated state and possibly a message. The goal of this algorithm is to create a session between the two parties and making it possible for both parties to calculate the shared secret key.

The algorithms are based on the ones used in Cohn-Gordon et al. but the *Initiate* and *Run* algorithm have an extra input *ephemeral* to show that in this algorithm an ephemeral key is used.

The state of the protocol that is used in the algorithms is a collection of variables. In order to access a single variable, an object-oriented approach is taken. For example, the notation to access variable v in the state is $:v$.

Definition 2. A state is a collection of variables. Both participants have a state of the protocol. The collection contains the following variables:

$:ephemeral \in \{0,1\}^{*}$, a random string for this stage, based on the security parameter. Used as the ephemeral key.

$:key$, is the key that is derived from the X3DH protocol.

$:otherid$, the identifier of the other party.

$:otherik$, the identity key of the other party.

$:otherspk$, the prekey of the other party.

$:role$, the role of the participant, it is either *initiator* or *responder*.

$:sessionid$, the session identifier. This contains all the public keys to which the participant has access.

$:status$, the status of the state. It can be either *empty*, *active*, *accepted* or *rejected*. *accepted* or *rejected* means that it has fully completed the computations and accepted or rejected the outcome.

We denote $^{i}_{x}$ as the state of session i of user x. In comparison to the state variables in Cohn-Gordon et al. we have removed some variables which are not necessary for our proof.

Combining the four algorithms gives an implementation of the X3DH protocol. The key generation algorithms create the keys that are needed for the protocol. The Initiate algorithm initiates the protocol. If the party that runs the Initiate algorithm has the role of responder, it outputs a prekey bundle. If the party that runs the Initiate algorithm has the role of initiator, it outputs nothing but receives the prekey bundle. The Run algorithm handles the rest of the algorithm. If the party that runs the Run algorithm has the role of initiator, it sends a message containing the ephemeral key EK to the responder. If the party that runs the Initiate algorithm has the role of responder, it outputs nothing but receives the message.

3.2.3 Modelling X3DH

Now we have to de ne the key indistinguishability experiment that is used in the proof. Before we give a formal de nition, we give an informal de nition as to provide a better understanding of the experiment. First the experiment establishes the identity keys or long-term secret keys before the adversary can interact with the model. The adversary has a lot of power in the model to try to break the protocol. It can learn some long-term secrets or even secrets in a single session with the help of reveal queries. These queries will reply the secret information that is requested, however, if too much is revealed it is clear the adversary can break the protocol. So we have to make sure that this will not be the case. This will be discussed in Section 3.2.5.

It is also possible for the adversary to let two parties start a session with chosen ephemeral keys. It can also control the delivery of these messages, for example by changing or dropping the message. If the adversary wants, it can start a test session, but it can do so only once. In that test session, the adversary is given a key, which is either the session key that corresponds with this session or a random key. It is then up to the adversary to decide if the key that it is given is the real key or a random key and if it guesses correctly it wins the experiment.

De nition 3. Let $$ be a key exchange protocol and $p; s; e \in \mathbb{N}$ as follows:

p is the number of parties,

s is the maximum number of sessions,

e is the maximum number of ephemeral or medium term keys for every party.

Let A be an algorithm. Then we de ne

$$\mathrm{Adv}_{;p;s;e}(A) = \left| 2\Pr\left[\mathrm{Exp}_{;p;s;e}(A) = 1\right] - 1 \right|; \tag{3.1}$$

where the experiment $\mathrm{Exp}_{;p;s;e}(A)$ is described in Algorithm 1. It internally calls the procedures of Algorithms 2, 3, 4, 5, 6 and 7. The algorithms are given in pseudocode, which is a non-formal coding language, used to make algorithms readable and understandable for the reader. We use the constructions `for` and `if` and some other notations:

$x \leftarrow y$ is used to assign a value y to a variable x.

$x \xleftarrow{\$} Y$ is used to randomly assign a value of Y to a variable x.

These algorithms use additional variables of the state that are defined below:

:prekeyid $\in \{1, \ldots, e\}$, which is the index of which prekey is used.

:otherprekeyid $\in \{1, \ldots, e\}$, which is the index of which prekey is used by the other party.

:ephemeral $\in \{0, 1\}$, which is the ephemeral private key that is used.

:reveal_ephemeral, a predicate that says if the reveal query for the ephemeral key is called.

:reveal_sessionkey, a predicate that says if the reveal query for the session key is called.

In comparison to the additional state variables in Cohn-Gordon et al. we have removed some variables which are not necessary for our proof.

3.2.4 Session Identifier

The state that we have defined is not specified in the Signal protocol itself. In our definition it holds all public keys involved in a specific session. We use the session identifier to restrict the adversary. If there is a session S which has the same identifiers as the test session, then the adversary can query the session key of S and use that to always succeed in the test session and the experiment. So we will limit the adversary in such a way that it cannot query a session which has the same session identifier as the test session.

The algorithms defined below are based on the algorithms from Cohn-Gordon et al. They have been altered to be used by the X3DH protocol.

Algorithm 1 $\mathrm{Exp}^{\mathrm{ind}}_{\Pi,p,s,e}(A)$

$b \xleftarrow{\$} \{0,1\}$
$\mathrm{tested} \leftarrow ?$
for $x = 1, \ldots, p$ **do** . Initialize the keys for each user
  $(IK_x, ik_x) \xleftarrow{\$} \mathrm{IdentityKeyGen}()$
  for $y = 1, \ldots, e$ **do** . Initialize signed prekeys for all sessions
    $(SPK^x_y, spk^x_y) \xleftarrow{\$} \mathrm{PreKeyGen}(ik_x)$
$\mathrm{public} \leftarrow (IK_1, \ldots, IK_p, SPK^1_1, \ldots, SPK^p_e)$
$b' \xleftarrow{\$} A^{\mathrm{Send},\mathrm{Reveal},\mathrm{Test}}(\mathrm{public})$. Test can only be called once
if $(\mathrm{tested} \neq ?) \wedge \mathrm{fresh}(\mathrm{tested}) \wedge b = b'$ **then** . Adversary was successful
  return 1
else . Adversary was not successful
  return 0

Algorithm 2 $\mathrm{Send}(x, i, m)$

if $\pi^i_x = ?$ **then** . There is no session yet
  $(\mathrm{peerid}, y, \mathrm{role}) \leftarrow m$
  $\pi^i_x.\mathrm{ephemeral} \xleftarrow{\$} \{0,1\}$
  $(\pi^i_x, m') \leftarrow \mathrm{Initiate}_{\pi^i_x.\mathrm{ephemeral}}(ik_x, spk^x_y, \mathrm{role}, \mathrm{peerid})$
  return m'
else . There is a session
  $(\pi^i_x, m') \leftarrow \mathrm{Run}_{\pi^i_x.\mathrm{ephemeral}}(ik_x, spk^x_y, \pi^i_x, m)$
  return m'

Algorithm 3 $\mathrm{RevealIdentityKey}(x)$

$\mathrm{revealed}.ik_x \leftarrow \mathrm{true}$
return ik_x

Algorithm 4 $\mathrm{RevealPreKey}(x, y)$

$\mathrm{revealed}.spk^y_x \leftarrow \mathrm{true}$
return spk^x_y

Algorithm 5 $\mathrm{RevealEphemeralKey}(x, i)$

$\pi^i_x.\mathrm{reveal\_ephemeral} \leftarrow \mathrm{true}$
return $\pi^i_x.\mathrm{ephemeral}$

Algorithm 6 $\mathrm{RevealKey}(x, i)$

$\pi^i_x.\mathrm{reveal\_sessionkey} \leftarrow \mathrm{true}$
return $\pi^i_x.\mathrm{key}$

Algorithm 7 Test(x, i)

```
1: if tested[x, i] ≠ ? or π_x^i.status ≠ accept then      . Test can only be called once
2:     return ?
3: tested[x, i] ← 1
4: if b = 0 then                                          . Return the real key
5:     return π_x^i.key
6: else                                  . Return a random string that looks like a key
7:     return random string
```

3.2.5 Predicates

With the current experiment it is possible for the adversary to break the protocol, by just revealing all secrets. We will define some predicates in order to limit this. If during the experiment, some of the predicates are not true, the adversary fails. If we now define these predicates in a correct way, we will have an adversary who cannot break the protocol by just revealing all secrets. For each predicate we will first provide an informal definition followed by a formal logical definition.

All predicates given in this section are from the paper of Cohn-Gordon et al. and only have slight modifications to them to only be used in the X3DH protocol.

First we will define the $\text{valid}(x, i)$ predicate, where x is the user and i is the index of a session of that user. A state is valid if it has fully completed the computations and it has accepted them and the session key is not queried by the adversary for this session or another session with the same session identifier. This gives us the following definition:

$$\begin{aligned}\text{valid}(x, i) = (&\pi_x^i.\text{status} = \text{accept}) \wedge \neg \pi_x^i.\text{reveal\_sessionkey} \\ &\wedge \forall j : \pi_x^i.\text{sessionid} = \pi_{\pi_x^i.\text{otherid}}^j.\text{sessionid} \\ &) : \neg \pi_{\pi_x^i.\text{otherid}}^j.\text{reveal\_sessionkey} :\end{aligned} \tag{3.2}$$

Now we will define the $\text{clean}(x, i)$ predicate. This predicate makes sure that the adversary does not reveal too many secret keys, so that it can calculate the session key. In order to calculate the session key in X3DH, one must calculate four Diffie-Hellman secret keys. Each of these Diffie-Hellman keys is calculated by a combination of personal keys of each party. For two parties Alice and Bob these combinations are shown in Figure 3.2. For each of these combinations, we will define a $\text{clean}_{XY}(x, i)$ predicate which states that the keys X and Y are not revealed by the adversary, where X is either the identity key, signed prekey or ephemeral key of Alice, and Y the same for Bob. We will denote by I the long-term identity key, M for the medium-term signed prekey and E for the ephemeral key. The $\text{clean}(x, i)$ predicate will be a disjunction of the four $\text{clean}_{XY}(x, i)$ predicates, because at least one must be true in order for the adversary to not be able to calculate

the session key. This gives us the following definition:

$$clean(x;i) = clean_{LM}(x;i) \_ clean_{EL}(x;i) \_ clean_{EM}(x;i) \_ clean_{EE}(x;i): \tag{3.3}$$

In all the subpredicates we need to make case distinctions on whether x is the initiator of the protocol or the responder. This is to define which key must not be revealed by the adversary. Thus, for the $clean_{LM}(x;i)$ predicate we see that in the case that x is the initiator, the identity key of x must not be revealed and the signed prekey of the other party must not be revealed. In the case that x is the responder, the identity key of the other party must not be revealed and the signed prekey of x must not be revealed.

$$clean_{LM}(x;i) = \begin{cases} : revealIk_x \wedge : revealSpk_{x.i_x.otherprekeyid} & i_x.role = initiator \\ : revealIk_{i_x.otherid} \wedge : revealSpk_{x.i_x.prekeyid} & i_x.role = responder: \end{cases} \tag{3.4}$$

Ephemeral Keys

With ephemeral keys we must first address a minor issue. We must be sure that the other party has generated an ephemeral key. In case of the initiator we must make sure the ephemeral key is generated by the responder. We do so by checking whether there exists a session for the responder, which we do by checking if the ephemeral key is included in the session identifier. We cannot compare the session identifiers, because the responder may not have responded yet, and as such has not generated a session identifier. If a session with the responder does not exist, then the ephemeral key would not be generated, or it could be generated by the adversary. We must also make sure that this ephemeral key is not revealed by the adversary in another session with the same session identifier.

In the case of the responder we already know that the initiator has generated an ephemeral key. However, we must make sure that both parties have the same session identifier and that the ephemeral key of the initiator is not revealed by the adversary in another session with the same session identifier.

We will note $a \sqsubset b$ as: "a is a substring of b". We will define a new subpredicate $clean_{Eph}(x;i)$ that will address this issue. For better readability we will define these predicates for the initiator first, and then for the responder.

Initiator

In the case that $i_x.role = initiator$, $clean^{init}_{Eph}(x;i)$ is defined as follows:

$$clean^{init}_{Eph}(x;i) = \exists j : OPK_j \sqsubset i_x.sessionid \wedge \forall j : OPK_j \sqsubset i_x.sessionid) : j \sqsubset i_{x.otherid}.reveal\_ephemeral : \tag{3.5}$$

Now we can define the rest of the $clean^{init}_{XY}(x, i)$ predicates for the initiator:

$$clean^{init}_{EL}(x, i) = \quad : \quad {}^{i}_{x}:reveal\_ephemeral \wedge : \quad reveal\_lk_{{}^{i}_{x}:otherid}; \tag{3.6}$$

$$clean^{init}_{EM}(x, i) = \quad : \quad {}^{i}_{x}:reveal\_ephemeral \wedge : \quad reveal\_spk_{{}^{i}_{x}:otherprekeyid}^{{}^{i}_{x}:otherid}; \tag{3.7}$$

$$clean^{init}_{EE}(x, i) = \quad : \quad {}^{i}_{x}:reveal\_ephemeral \wedge clean^{init}_{Eph}(x, i): \tag{3.8}$$

Responder

In the case that ${}^{i}_{x}:role = responder$, $clean^{resp}_{Eph}(x, i)$ is defined as follows:

$$\begin{aligned} clean^{resp}_{Eph}(x, i) = \; & 9j \; : \; {}^{i}_{x}:sessionid = {}^{j}_{{}^{i}_{x}:otherid}:sessionid \\ \wedge \; & 8j \; : \; {}^{i}_{x}:sessionid = {}^{j}_{{}^{i}_{x}:otherid}:sessionid \\ &) : \quad {}^{j}_{{}^{i}_{x}:otherid}:reveal\_ephemeral \; : \end{aligned} \tag{3.9}$$

Now we can define the rest of the $clean^{resp}_{XY}(x, i)$ predicates for the responder:

$$clean^{resp}_{EL}(x, i) = clean^{resp}_{Eph}(x, i) \wedge : \; reveal\_lk_{x}; \tag{3.10}$$

$$clean^{resp}_{EM}(x, i) = clean^{resp}_{Eph}(x, i) \wedge : \; reveal\_spk_{x}^{{}^{i}_{x}:prekeyid}; \tag{3.11}$$

$$clean^{resp}_{EE}(x, i) = clean^{resp}_{Eph}(x, i) \wedge : \quad {}^{i}_{x}:reveal\_ephemeral: \tag{3.12}$$

Now we can define the $fresh(x, i)$ predicate. This predicate states that a session is valid and that not too many keys have been revealed by the adversary. So this will be the the combination of the $valid(x, i)$ and $clean(x, i)$ predicates:

$$fresh(x, i) = valid(x, i) \wedge clean(x, i): \tag{3.13}$$

Break Event

We denote the event $break_i$ as the event that the adversary A wins Game i. We denote Adv_i as the advantage of A against Game i and is defined as follows:

$$Adv_i := |2 \; Pr(break_i) \quad 1|: \tag{3.14}$$

3.3 Proof

3.3.1 Scope of the Proof

There are some aspects of the X3DH protocol that we did not model and will not prove. One of these aspects is the different implementations of the X3DH protocol used by messaging services. We focus on the protocol as documented by the developers of X3DH. We also do not make any assumptions on the underlying primitives that are used, for example what elliptic curve is used.

We do not prove all the security goals that Signal claims X3DH has, such as plausible deniability, instead, we focus on one-to-one messaging. This means we do not look at group messaging or other functionalities that can be offered.

As explained in Section 3.2.1 we do not assume side-channel attacks.

There are also some technicalities we have to discuss.

For the Test algorithm we will consider the party that is used in the Test algorithm to be the initiator of the session. We will leave the proof in case of the responder as analogous.

We assume that the Key Derivation Function that is used is a random oracle, so we do not have to make any assumptions on the KDF.

3.3.2 Proof of Security

With the help of game hopping we can now provide a proof of security. The goal is to prove that Adv_0 is bounded by the Gap Dif e-Hellman hardness assumption (See Section 2.6.4). In the X3DH protocol there are four keys that need to be kept a secret. This means that we have to consider four cases in the proof. These cases are somewhat similar, so we will give a proof for a speci c case and for the other cases we will discuss brie y the differences and how they impact the outcome.

First we will construct ve games that affect all four cases, after which we will construct four games for each case. An overview of the proof is shown in Figure 3.3. In the last game for each case the shared secret key will be replaced by a random value. Then we can calculate a bound on the success probability of the adversary. We will construct each game towards a reduction to Gap Dif e-Hellman (GDH). This means that we will replace some Dif e-Hellman keys with values that we query from a GDH oracle. The games that are used in the proof are from the proof given by Cohn-Gordon et al. [11]. The games are rewritten to be used with the X3DH protocol. Game 3 is different than the games that were de ne by Cohn-Gordon et al. and is our contribution to the proof.

The entity that the adversary sends his messages to is called the challenger, which is often denoted as C. This entity "challenges" the adversary to break the system.

Theorem 2. The Extended Triple Dif e-Hellman (X3DH) protocol used in the Signal protocol is a secure key exchange protocol under the Gap Dif e-Hellman assumption and assuming the Key Derivation Function is a random oracle.

Proof. We start with constructing ve games which are used in all four cases.

Next to the variables de ned in Section 3.2.3 we will de ne a new variable, q, which is the order of the group that we are working in. We will recall the previous variables that we will use in the proof:

p is the number of parties,

s is the maximum number of sessions,

e is the maximum number of ephemeral or medium term keys for every party.

Figure 3.3: An overview of the proof. Most cases are similar to each other, so in the proof we handle one case and explain the differences in the other cases.

Game Hops for All Cases

Game 0: The initial experiment

This is the experiment we have defined in Section 3.2.3. The advantage of the adversary $\mathcal{A}$ against this game is Adv_0.

Game 1: No Diffie-Hellman collisions

In order to help us in further game hops, we will first make sure that there are no Diffie-Hellman public keys that are the same. Having two keys that have the same value is also called a collision. To be able to make sure that there are no collisions, the challenger $\mathcal{C}$ will keep a list containing all the Diffie-Hellman private keys. This way it is easy to check for $\mathcal{C}$ if a value appears twice. If this is the case, the game is aborted and the adversary loses. Each party has generated a long-term identity key and a maximum of e ephemeral keys. This gives us a total of $p + ep$ Diffie-Hellman keys and a total of $\binom{p+ep}{2}$ pairs of keys. Every single pair of these keys must not collide. Two keys in the same group with order q will collide with a probability of $\frac{1}{q}$. Now we can give the following bound on the advantage of the adversary:

$$\mathrm{Adv}_0 \leq \frac{\binom{p+ep}{2}}{q} + \mathrm{Adv}_1. \tag{3.15}$$

Game 2: Guess test session

In this game, the challenger must guess which session will be tested by the adversary with $\mathsf{Test}(x, i)$. This means that $\mathcal{C}$ must guess both x and i correctly. For x there are a total of p possibilities, and for i there are s possibilities. That means that the probability that the challenger guesses both of them correctly is $\frac{1}{p \cdot s}$. This will be a game hop with a large failure event. $\mathcal{C}$ guesses $(x^*, i^*) \in [1 \ldots p] \times [1 \ldots s]$. We will define an event E where $\mathcal{A}$ will query $\mathsf{Test}(x, i)$, where $(x, i) \neq (x^*, i^*)$. We will abort the game when E occurs. The probability that E occurs is $\frac{1}{p \cdot s}$. From this we get the following:

$$\mathsf{Adv}_1 \geq p \cdot s \cdot \mathsf{Adv}_2. \tag{3.16}$$

Game 3: Guess unique partner session

We must also guess the partner session that is tied to the test session. Since in Game 1 we made sure that all the Diffie-Hellman keys are unique and as such the combination of identity keys, we know that the partner session is also unique. This closely follows Game 2, as in Game 2 we guessed a certain session as well. This is a game hop with a large failure event. $\mathcal{C}$ guesses a pair $(x^*, i^*) \in [1 \ldots p] \times [1 \ldots s]$. We will define an event E where the partner session is $\hat{s}^{i^*}_{x^*}$ where $(x, i) \neq (x^*, i^*)$. Since the partner session is unique, we know that the probability of $\neg E$ occurring is $\frac{1}{p \cdot s}$.

The session identifiers contain the public keys. However, since the adversary can pick ephemeral keys, it is possible that there is another session that has the same identifier as the test session that is not this partner session. We will make sure that if such a session exists, then the challenger knows about it. This is another hop with a large failure event. We do this by guessing the index $j \in \{1 \ldots s\}$ of this session. We will define an event F where this session is $\hat{s}^j_{x^*}$ and $i \neq j$. The probability that $\neg F$ occurs is $\frac{1}{s}$. Combining both failure events gives us the following bound:

$$\mathsf{Adv}_2 \geq p \cdot s^2 \cdot \mathsf{Adv}_3. \tag{3.17}$$

Game Hops for Specific Cases

Now we have to move to a game distinction. We can now write the advantage of $\mathcal{A}$ in Game 3 as follows:

$$\mathsf{Adv}_3 \leq \underbrace{\mathsf{Adv}_3^{\mathsf{clean}_{\mathsf{LM}}(x,i)}}_{\text{Case 1}} + \underbrace{\mathsf{Adv}_3^{\mathsf{clean}_{\mathsf{EL}}(x,i)}}_{\text{Case 2}} + \underbrace{\mathsf{Adv}_3^{\mathsf{clean}_{\mathsf{EM}}(x,i)}}_{\text{Case 3}} + \underbrace{\mathsf{Adv}_3^{\mathsf{clean}_{\mathsf{EE}}(x,i)}}_{\text{Case 4}}. \tag{3.18}$$

Where $\mathsf{Adv}_3^{\mathsf{clean}_{XY}(x,i)}$ is the the advantage of the adversary in the case that $\mathsf{clean}_{XY}(x, i)$ must stay true.

We will define all the game hops for the case $\mathrm{Adv}_3^{\mathrm{clean}_{LM}(x;i)}$ and $\overset{i}{x}\text{:role} = \text{initiator}$. As was mentioned before, the case of $\overset{i}{x}\text{:role} = \text{responder}$ is analogous, and only one of the two appears. The cases for the $\mathrm{clean}_{LM}(x;i)$ predicates will be given later, but since they are very similar, we will only give the differences.

Case 1: $\mathrm{clean}_{LM}(x;i)$

In this case we must make sure that $\mathrm{clean}_{LM}(x;i)$ stays true. This means that A cannot have issued the queries $\mathrm{RevealIdentityKey}(x)$ and $\mathrm{RevealPreKey}(n;y)$ where $\overset{i}{x}\text{:otherprekeyid} = n$.

Game 4: Guessing the prekey of the other party

In this game the challenger has to guess the index of the signed prekey that is used by the other party that is used in the Test session. C will guess $n \in \{1,\ldots,e\}$. This is a game hop with a large failure event. We will define an event E where n is not the index of the signed prekey that is used. The probability of E is $\frac{1}{e}$, which gives us the following:

$$\mathrm{Adv}_3 \leq e\ \mathrm{Adv}_4. \tag{3.19}$$

Game 5: Allowing some duplicate values

To prepare for future game hops, we will allow the identity key of party x, IK_x, and the signed prekey of party y with index n which we guessed in Game 4, SPK_n^y, to be identical. We do this because we want to introduce a GDH challenger. With this challenger these keys are allowed to be the same. We are working in a group of order q, so the probability that the keys are the same is $\frac{1}{q}$. This is a game hop with a small failure event and gives us the following:

$$\mathrm{Adv}_4 \leq \frac{1}{q} + \mathrm{Adv}_5. \tag{3.20}$$

Game 6: Introducing GDH

In order to calculate the shared secret key of the X3DH protocol, we need four parts to be used as input to the KDF. We are in the case $\mathrm{clean}_{LM}(x;i)$, which means that ik_x and spk_n^y remained a secret to the adversary. The shared secret key is calculated as follows: $K_{AB} = \mathrm{KDF}(K_1 || K_2 || K_3 || K_4)$. Without loss of generality A knows K_2, K_3 and K_4 and it wins if it queries $\mathrm{KDF}(K_1 || K_2 || K_3 || K_4)$. So in order for A to win, A must find $K_1 = g^{ik_x\ spk_n^y}$ for the generator g of the group. We will

abort this game when $\mathcal{A}$ queries K_1 to the KDF oracle it has access to, because that is the case that the protocol is broken. We will define this event E_{break}. We will show that if E happens, the adversary will win the game, which we have defined as $\text{Adv}(\text{break}_6)$. So we can split the advantage as follows:

$$\text{Adv}_5 \quad \text{Adv}(\text{break}_6) + \text{Adv}_6: \tag{3.21}$$

We will now show that in the case of E, we can create an algorithm $\mathcal{A}$ that can win the GDH problem. As was explained in Section 2.6.4, in the GDH problem $\mathcal{A}$ is given a triple $(g; g^a; g^b)$, and its goal is to find $K = g^{ab}$ with access to a DDH oracle.

Now $\mathcal{A}$ will replace IK_x with g^a and SPK_n^y with g^b, with both a and b unknown to $\mathcal{A}$. $\mathcal{A}$ will then simulate Game 5. Since IK_x and SPK_n^y are replaced, it is not possible to calculate some session keys between certain parties. Since the adversary $\mathcal{A}$ can set up sessions between parties, $\mathcal{A}$ must simulate these calculations by giving random keys. These sessions are:

1. A session which is not the Test session that is between party x and party y, where x is the initiator. Since g^a and g^b are both used to calculate the session key, this would require knowledge about a or b.

2. A session which is not the Test session that is between party x and any other party, including y, where x is the responder. Since g^a is used and some other ephemeral public key on which we have no information, this would require knowledge about a.

3. A session which can be the Test session that is between any party that is not x and party y, where y is the responder. Since g^b is used and some other ephemeral public key on which we have no information, this would require knowledge about b.

$\mathcal{A}$ will keep a list which contains, for each session for which a random key is used, the random key and which public keys should have been used for the calculations. By keeping this list, $\mathcal{A}$ must be consistent in answering queries from $\mathcal{A}$ by giving the right random keys for the right sessions. Before simulating a session and picking a random key, it is also important that $\mathcal{A}$ at will use the DDH oracle to check if a previous random oracle query matches the session, in order to give the right key. Otherwise, the simulation will not be consistent.

It is also important that $\mathcal{A}$ will not answer the reveal queries which will violate the clean predicate, as this predicate must stay true or the adversary loses.

If $\mathcal{A}$ queries the KDF oracle with a query of the form $g^{z_1} || g^{z_2} || g^{z_3} || g^{z_4}$, then $\mathcal{A}$ must first go through its list and use the DDH oracle to check if the public keys match the corresponding g^{z_i}, for $i \in \{1 : : : 4\}$. These g^{z_i} each match the K_i that is used to calculate the shared secret key. We refer to Figure 3.2 to see which keys are needed to calculate which K_i. For each session type that uses random keys we can then construct DDH oracle queries to give us the information we need.

For sessions of type 1, A has no knowledge about b and some other ephemeral private key e, so in that case A will query the DDH oracle $(g; g^a; g^b; g^{z_1})$ to see if the first key is correct, and $(g; g^e; g^b; g^{z_3})$ to see if the third key is correct. A knows enough to check the other keys itself.

For sessions of type 2, A has no knowledge about a and e, which is some other ephemeral private key. In that case we will query the DDH oracle $(g; g^a; g^e; g^{z_2})$, in order to see if the second key is correct. A knows enough to check the other keys itself.

For sessions of type 3, A has no knowledge about b and e, which is some other ephemeral private key. In that case we will query the DDH oracle $(g; g^b; g^e; g^{z_3})$, in order to see if the third key is correct. A knows enough to check the other keys itself.

For each case, if the DDH oracle returns 1, meaning that the key is correct, then A can use the random keys that it generated from the list it keeps. If the oracle does not return 1, then A must generate a new random value and query that value to the oracle.

It is important to note that if A queries $(g; g^a; g^b; g^{z_1})$ to the DDH oracle, and it returns 1, then A has found the solution to the GDH problem, and will just return g^{z_1} as the answer to the GDH challenger. In other words, if A breaks Game 6, then A breaks the GDH problem.

So from this we have that if E occurs, then we have a solution to the GDH problem. This gives us the following:

$$\mathrm{Adv}(\mathrm{break}_6) \quad {}_{\mathrm{GDH}}(A): \tag{3.22}$$

Where ${}_{\mathrm{GDH}}(A)$ is the probability of guessing the right answer for the GDH problem.

Game 7: Replacing the session key

The last game is where the adversary tries to find the session key. However, before that happens, the experiment will generate a randomly chosen string that is from the same space that all the keys in this protocol come from. This means that it is not detectable by the adversary whether the keys have been replaced. We know from Game 6 that A did not query the KDF oracle with the correct input. The session key is now replaced with a random other key, depending on the bit that was randomly chosen at the start of the experiment. The adversary will have no advantage anymore, which gives us the following:

$$\mathrm{Adv}_6 = \mathrm{Adv}_7 = 0: \tag{3.23}$$

Of all these games combined in case 1, we can derive the following:

$$\mathrm{Adv}_3^{\mathrm{clean}_{\mathrm{LM}}(x;i)} \quad e \quad \frac{1}{q} + {}_{\mathrm{GDH}}(A) \quad : \tag{3.24}$$

Case 2: $\text{clean}_{EL}(x, i)$

In this case we make sure that the $\text{clean}_{EL}(x, i)$ predicate stays true. This means that A cannot have issued the queries $\text{RevealEphemeral}(x, i)$ and $\text{RevealIdentityKey}(y)$ in the case that $\pi_x^i.\text{role} = \text{initiator}$ and $\pi_x^i.\text{otherid} = y$ is the other party of the session. In the case that $\pi_x^i.\text{role} = \text{responder}$ the queries $\text{RevealEphemeralKey}(y, i)$ and $\text{RevealIdentityKey}(x)$ cannot have been issued.

The differences from the previous games is that we no longer have to guess the index in Game 4. In Game 5 we allow EK_x and IK_y to be duplicate, and in Game 6 we replace these keys with the GDH values g^a and g^b. From this we get the following:

$$\text{Adv}_3^{\text{clean}_{EL}(x,i)} \quad \frac{1}{q} + \quad_{GDH}(A): \tag{3.25}$$

Case 3: $\text{clean}_{EM}(x, i)$

In this case we make sure that the $\text{clean}_{EM}(x, i)$ predicate stays true. This means that A cannot have issued the queries $\text{RevealEphemeral}(x, i)$ and $\text{RevealPreKey}(n, y)$, where $\pi_x^i.\text{otherprekeyid} = n$ and $\pi_x^i.\text{role} = \text{initiator}$. For $\pi_x^i.\text{role} = \text{responder}$ the queries $\text{RevealEphemeralKey}(y, i)$ and $\text{RevealPreKey}(\pi_x^i.\text{prekeyid}, x)$ cannot have been issued.

In this case we do have to guess the index again in Game 4. We will allow EK_x and SPK_y to be duplicate in Game 5. In Game 6 we will replace these keys with the GDH values. From this we get the following:

$$\text{Adv}_3^{\text{clean}_{EM}(x,i)} \quad e \quad \frac{1}{q} + \quad_{GDH}(A) \quad : \tag{3.26}$$

Case 4: $\text{clean}_{EE}(x, i)$

In this case we make sure that the $\text{clean}_{EE}(x, i)$ predicate stays true. This means that A cannot have issued the queries $\text{RevealEphemeral}(x, i)$ and $\text{RevealEphemeralKey}(y, i)$ in both the cases that $\pi_x^i.\text{role} = \text{initiator}$ and $\pi_x^i.\text{role} = \text{responder}$.

In this case we guess the index again in Game 4. However, this time we do not incur a factor of e but a factor of s. This is because ephemeral keys are generated on a session basis and not beforehand. In Game 5 we allow the ephemeral keys of both parties to be the same, and in Game 6 we replace these keys with the GDH values. From this we get the following:

$$\text{Adv}_3^{\text{clean}_{EM}(x,i)} \quad s \quad \frac{1}{q} + \quad_{GDH}(A) \quad : \tag{3.27}$$

Combining the four cases

We see that all cases are bounded by the GDH problem. Now we can combine all the cases to get the following bound on Game 3:

$$\begin{aligned} \mathrm{Adv}_3 \quad & e\left(\frac{1}{q}+\ \right)_{\mathrm{GDH}}(A\ ) \quad +\left(\frac{1}{q}+\ \right)_{\mathrm{GDH}}(A\ )+e\left(\frac{1}{q}+\ \right)_{\mathrm{GDH}}(A\ ) \\ & +s\left(\frac{1}{q}+\ \right)_{\mathrm{GDH}}(A\ )\ : \end{aligned} \tag{3.28}$$

From this equation we can derive a bound for Game 0, the original experiment:

$$\begin{aligned} \mathrm{Adv}_0 \quad & \frac{\frac{p+ep}{2}}{q}+p^2\ \ s^3 \quad e\left(\frac{1}{q}+\ \right)_{\mathrm{GDH}}(A\ ) \quad +\left(\frac{1}{q}+\ \right)_{\mathrm{GDH}}(A\ ) \\ & +e\left(\frac{1}{q}+\ \right)_{\mathrm{GDH}}(A\ ) \quad +s\left(\frac{1}{q}+\ \right)_{\mathrm{GDH}}(A\ ) \\ & \frac{\frac{p+ep}{2}}{q}+p^2\ \ s^3 \quad \frac{2e+s+1}{q}+(2\ \ e+s+1)\ _{\mathrm{GDH}}(A\ )\ : \end{aligned} \tag{3.29}$$

□

Chapter 4

Conclusions

In this thesis, we have considered X3DH in the Signal protocol, and derived a proof of security. From the proof it follows that X3DH provides secrecy and authentication.

We proved this by showing that a powerful adversary with control over the network cannot calculate the shared secret key to expose the messages that are being sent. This even holds if the adversary has access to some of the secret values that are used to calculate the shared secret key. We proved it by showing that the advantage that an adversary has in breaking the X3DH protocol is bounded by the Gap Dif e-Hellman hardness assumption.

One could argue that the signatures of prekeys are not necessary, however because of these signatures, it is not possible for a malicious server to provide forged prekeys to a party and then reveal the communication between two parties. This signature shows that these keys were published by one of the parties and not by the server itself [29].

While doing the literature research on this topic, it appeared that existing research on the X3DH protocol was slim. It has been discussed in various papers as part of the Signal protocol, but it is not looked at on its own.

4.1 Future Work

As was brie y discussed in Section 3.3.1, there are some parts of the X3DH protocol we did not include in our proof but are possible starting points for future research. As this is a protocol which is widely used, new features will continue to be developed and added to implementations of the protocol. These would deserve future investigation.

We did not handle out-of-order decryption, which requires users to store secret values until the message has been delivered. It is not yet clear how much impact storing these secret values have, but it could reduce the forward secrecy that the X3DH protocol provides.

While we have proven some security goals of the X3DH protocol, there are

some that are still not proven as of yet, for example plausible deniability. While this may not be as impactful as secrecy, it is still a security goal and as such would require a proof.

Each implementation of the Signal protocol differs, and as such, different implementations of the X3DH protocol are used. We only proved the security of X3DH protocol as documented by the developers. Therefore, the proof that is given might not hold for other implementations of the X3DH protocol.

We assumed in our proof that the public keys were veri ed before they were used in the protocol. In the real world, this is not always the case, and with no veri cation of the keys on a different channel of communication, it is possible for an adversary to impersonate other parties. It is dif cult to make sure that public keys are veri ed, and so far there is no real way to provide this veri cation by not using another channel of communication.

Bibliography

[1] M. Hamburg A. Langley and S.Turner. Elliptic Curves for Security. Internet Engineering Task Force; RFC 7748 (Informational); IETF, January 2016. urlhttps://www.ietf.org/rfc/rfc7748.txt.

[2] N. Z. Almuzaini and I. Ahmad. Formal Analysis of the Signal Protocol Using the Scyther Tool. In2019 2nd International Conference on Computer Applications Information Security (ICCAIS), pages 1–6, 2019.

[3] Feng Bao, Robert H Deng, and Huafei Zhu. Variations of Dif e-Hellman problem. InInternational conference on information and communications security, pages 301–312. Springer, 2003.

[4] Boaz Barak and Shai Halevi. A model and architecture for pseudo-random generation with applications to /dev/random. Cryptology ePrint Archive, Report 2005/029, 2005. https://eprint.iacr.org/2005/029 .

[5] Mihir Bellare and Phillip Rogaway. Entity Authentication and Key Distribution. In Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '93, page 232–249, Berlin, Heidelberg, 1993. Springer-Verlag.

[6] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing ef cient protocols. InProceedings of the 1st ACM conference on Computer and communications security, pages 62–73, 1993.

[7] Mihir Bellare and Phillip Rogaway. Code-Based Game-Playing Proofs and the Security of Triple Encryption. Cryptology ePrint Archive, Report 2004/331, 2004. https://eprint.iacr.org/2004/331 .

[8] Dan Boneh. The decision dif e-hellman problem. InInternational Algorithmic Number Theory Symposium, pages 48–63. Springer, 1998.

[9] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. InInternational Conference on the Theory and Applications of Cryptographic Techniques, pages 255–271. Springer, 2003.

[10] Rein Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In *Annual International Cryptology Conference*, pages 90–104. Springer, 1997.

[11] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A Formal Security Analysis of the Signal Messaging Protocol. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 451–466, 2017.

[12] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. Post-Compromise Security. Cryptology ePrint Archive, Report 2016/221, 2016. https://eprint.iacr.org/2016/221 .

[13] Cas Cremers, Jaiden Fairoze, Benjamin Kiesl, and Aurora Naska. Clone Detection in Secure Messaging: Improving Post-Compromise Security in Practice. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1481–1495, New York, NY, USA, 2020. Association for Computing Machinery.

[14] K. Igoe D. McGrew and M. Salter. Fundamental Elliptic Curve Cryptography Algorithms. Internet Engineering Task Force; RFC 6090 (Informational); IETF, February 2011. urlhttps://www.ietf.org/rfc/rfc6090.txt.

[15] Bert Den Boer. Dif e-Hellman is as strong as discrete log for certain primes. In *Conference on the Theory and Application of Cryptography*, pages 530–539. Springer, 1988.

[16] Alexander W. Dent. A Note On Game-Hopping Proofs. Cryptology ePrint Archive, Report 2006/260, 2006. https://eprint.iacr.org/2006/260 .

[17] Whit eld Dif e and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[18] Facebook. Messenger Secret Conversations Technical Whitepaper, July 2016. https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf .

[19] A. Behrouz. Forouzan. *Data communications & networking Fourth Edition*, pages 952–956. McGraw-Hill Education, fourth edition, 2006.

[20] A. Behrouz. Forouzan. *Data communications & networking Fourth Edition*, pages 971–976. McGraw-Hill Education, fourth edition, 2006.

[21] Stefan Friedl. An elementary proof of the group law for elliptic curves. *Groups Complexity Cryptology*, 9(2):117–123, 2017.

[22] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[23] N. Kobeissi, K. Bhargavan, and B. Blanchet. Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 435–450, 2017.

[24] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[25] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). Internet Engineering Task Force; RFC 5869 (Informational); IETF, May 2010. urlhttps://www.ietf.org/rfc/rfc5869.txt.

[26] Hugo Krawczyk. Cryptographic Extraction and Key Derivation: The HKDF Scheme. Cryptology ePrint Archive, Report 2010/264, 2010. `https://eprint.iacr.org/2010/264`.

[27] Joshua Lund. Signal partners with Microsoft to bring end-to-end encryption to Skype. `https://signal.org/blog/skype-partnership/`.

[28] Moxie Marlinspike and Trevor Perrin (editor). The Double Ratchet Algorithm, November 2016. `https://signal.org/docs/specifications/doubleratchet/`.

[29] Moxie Marlinspike and Trevor Perrin (editor). The X3DH Key Agreement Protocol, November 2016. `https://signal.org/docs/specifications/x3dh/`.

[30] Kevin S McCurley. The discrete logarithm problem. In *Proc. of Symp. in Applied Math*, volume 42, pages 49–74. USA, 1990.

[31] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *Public Key Cryptography*, pages 104–118, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[32] E. Rescorla. Diffie-Hellman Key Agreement Method. Network Working Group; RFC 2631; IETF, June 1999. urlhttps://www.ietf.org/rfc/rfc2631.txt.

[33] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. Internet Engineering Task Force; RFC 8446; IETF, August 2018. urlhttps://www.ietf.org/rfc/rfc8446.txt.

[34] P. Rösler, C. Mainka, and J. Schwenk. More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 415–429, 2018.

[35] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. https://eprint.iacr.org/2004/332.

[36] Joseph H Silverman and Joe Suzuki. Elliptic curve discrete logarithms and the index calculus. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 110–125. Springer, 1998.

[37] WhatsApp. WhatsApp Encryption Overview Technical White Paper, December 2017. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf.