Analysis

Security of Ibex

Paul Gerhart, Paul Rösler & Dominique Schröder

July 30, 2023

Contents

1 Abstract

This document delves into a comprehensive analysis of the security features of Threema's novel cryptographic protocol, known as Ibex. The analysis was conducted by a team of experts comprising Paul Gerhart, Paul Rösler, and Dominique Schröder. To ensure the accuracy and completeness of their evaluation, the Threema team provided the researchers with the necessary specifications and actively addressed any queries that arose during the analysis.

The evaluation of the Ibex protocol was carried out independently, adhering to the provable security approach, which serves as the gold standard in the field. By following this rigorous methodology, the researchers aimed to assess the robustness and reliability of Threema's security measures.

The document provides a comprehensive analysis of Threema's Ibex protocol in a structured manner. It introduces the messenger and establishes the context for evaluation. It explains the basics of cryptography, including number theory and the communication model. The Ibex functionality is formalized, focusing on its key components and operations in Section 4. The document then addresses the security properties aimed to be achieved by the Ibex protocol in Section 5. It outlines specific goals and requirements for ensuring high security and privacy for Threema users. Section 6 offers a detailed formalization of the Ibex protocol, covering its design, algorithms, and secure communication mechanisms. Section 7 presents the security proof for the Ibex protocol. Moving on, Section 8 explores practical instantiations of the cryptographic primitives used in the Ibex protocol. Lastly, Section 9 provides recommendations for future development and enhancement of the Ibex protocol.

1.1 Summary of the Findings

The security analysis covers various important properties to ensure the robustness of the Ibex protocol. The analysis includes an assessment of confidentiality, which focuses on preserving authorized restrictions on information access and disclosure and protecting personal privacy and proprietary information. The analysis also examines authenticity, guaranteeing that the origin of operations or data cannot be tampered with. Lastly, forward security is evaluated, ensuring that past communications remain secure even if a party's long-term secret keys are compromised in the future. Our formal security analysis did not reveal any flaws in the design of Ibex.

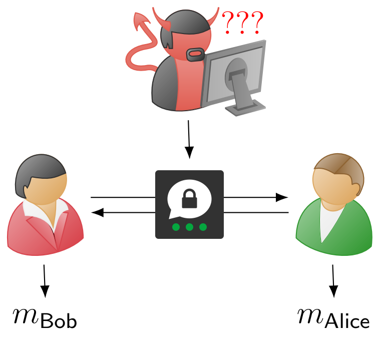
More specifically, we show that an efficient attacker capable of breaking one of the security properties of the Ibex protocol would also break the Gap-Diffie-Hellman (GDH) problem. Since it is assumed that the GDH problem is hard, which means it cannot be broken by any adversary in polynomial time, there can be no efficient adversary against Ibex.

2 Introduction

This section provides an introduction to messengers and their benefits compared to traditional forms of communication. Additionally, we provide a high-level overview of key exchange protocols and secure channels, which are the main building block in secure messaging protocols.

2.1 Introduction to Messengers

A messenger is a software or application that enables users to exchange messages in real-time. It can be used for personal or professional purposes and can include various features such as text messaging, voice and video calls, file sharing, and more.



Modern messengers offer several benefits compared to traditional forms of communication like email, SMS, or phone calls. These benefits include:

Privacy A secure messenger can protect message metadata from being collected and analyzed by third parties, including sender and receiver information, location, and timestamps.

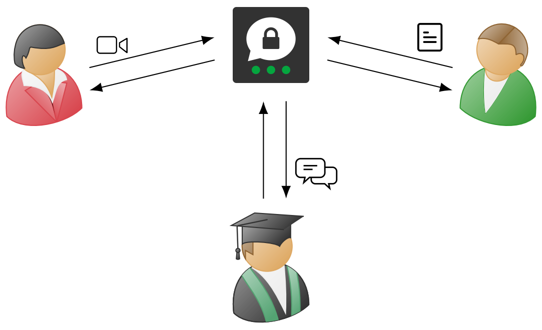
End-to-End Encryption Modern messengers use end-to-end encryption (E2EE) to secure messages and other data. This ensures that only the sender and recipient can read the content of the message, and no one else, including the messenger service provider, can access it.

Secure Authentication Messengers use secure authentication protocols to verify user identities, ensuring that only authorized users can access the service and exchange messages.

Messengers rely on a secure messaging protocol (SMP) to achieve these benefits. An SMP is a set of procedures that utilize cryptographic schemes to send and receive messages (and other data types like files, etc.). An SMP protocol outlines the creation, distribution, authentication, usage, and deletion of cryptographic keys, message preparation (encryption, authentication, etc.) for transmission, and the processing of received packets (decryption, verification, etc.).

2.2 High-Level Overview of the Cryptographic Goals of Threema

In this section, we give a high-level overview of the cryptographic goals of Threema. In a messaging application, multiple users wish to interact with each other securely. Interacting means that the users wish to send and receive chat messages, send files, or have a call using the messaging application. While calls are clearly only feasible if both users are online, sending files or receiving chat messages should also be possible if the other communication party is offline. To enable this asynchronous communication, the communicating parties interact with a central server instead of directly talking to each other. This server can be assumed to be online all the time, and therefore, asynchronous communication is possible. In a perfect world, we trust this central server completely, and now the communication problem would be solved. However, in practice, we wish to achieve security and privacy goals without relying on the trusted central party.



To achieve these goals, we will leverage the benefits of availability given by the central server with those of private communication obtained by modern cryptographic techniques. Cryptographic techniques are used to realize confidentiality, integrity, and authenticity. Confidentiality ensures that the communication remains private and inaccessible to third parties. We can achieve confidentiality by encrypting the exchanged data using a secret key that is exchanged between the communicating parties. Meanwhile, integrity guarantees that the communication remains unaltered and cannot be tampered with by any unauthorized third party. We can achieve integrity by making our encrypted data tamper-resistant using a message authentication code. Lastly, authenticity ensures that the communicating parties can always be certain of the identity of the other party involved. We can achieve authenticity by exchanging public key pairs. In a synchronous setting, this blueprint may lead to a decent messaging protocol. However, we want our messaging application to allow asynchronous communication and enable communication between all users. Therefore, we have to find a way to exchange secret keys between users that have never talked to each other in an authenticated manner such that any eavesdropping party can not learn the secret key. Even if this requirement seems to be impossible, cryptographic key exchange protocols satisfy it. We give a more formal description of the used functionality in Section 4 and an overview of the security goals in Section 5.

3 Basics

In this section, we introduce the cryptographic fundamentals used in the technical details presented in this work. These basics include provable security, mathematical foundations, and cryptographic hardness assumptions. The formal description of these fundamentals is crucial for the verifiability of the technical aspects of the subsequent sections.

3.1 Provable Security

This section introduces the concept of provable security, the de facto standard for modern security analysis. First, we describe the formalization of cryptographic schemes in Section 3.1.1. This formalization involves the description of the interfaces, which is discussed in Section 3.1.2. Our analysis formalizes the security w.r.t. cryptographic games, which we introduce in Section 3.1.3. The security of cryptographic schemes is reduced to some hardness assumption, which means that a security breach in our model results in an algorithm that breaks some underlying hard problems. We discuss this technique in Section 3.1.4.

3.1.1 Formalizing Cryptographic Schemes and Protocols

The formalization of cryptographic schemes consists of three steps. The first step formalizes what the cryptographic scheme does, which is typically done in the formal *description of the functionality.* The cryptographic functionality specifies the *interfaces*, which formalizes the individual algorithms that realize the functionality. As an example, let us consider an encryption scheme. Intuitively, an encryption scheme converts something readable, the plaintext, into an object that does not leak any (non-trivial) information about the original text. We call this output the ciphertext. While this describes the high-level functionality of the encryption algorithm, it does not tell us if an encryption involves more algorithms. It also makes no statement about the inputs of these algorithms. The formal definition of a cryptographic primitive can be seen as the definition of a class in any programming language. The formalization of the interface then concludes with the *definition of the correctness* that describes the behavior, i.e., it formalizes the *honest* execution of the actual scheme.

The second part then consists of the *formalization of the security properties*, which is typically done as a *cryptographic game* (see Section 3.1.3). A cryptographic game is a mind experiment between an adversary and a challenger. Intuitively, the adversary is attacking the cryptographic scheme, and the challenger provides the interfaces for the adversary. Moreover, once the adversary finishes its attack by returning some information that serves as an indication for breaking the scheme, the challenger evaluates the output and indicates whether it corresponds to a valid attack. The possibilities of the attacker in the cryptographic game must match as close as possible to its capabilities in the real world. If the model is too weak, then the attacker might execute an attack that the model does not cover. And because it might not be covered, it might result in cryptographic schemes that are vulnerable against that attack and, therefore, insecure from a practical perspective. On the other hand, if the model is too strong, then we might have the problem that either no cryptographic scheme exists that satisfies this definition, or it might be too inefficient and, therefore, not be usable in practice.

The third part consists of a *cryptographic construction* and its *security* evaluation. The cryptographic construction is an algorithmic instantiation of the individual interfaces. In other words, it precisely says which computations must be performed to realize the interfaces. Finally, the security evaluation shows that the cryptographic scheme is secure with respect to the security model. In many cases, proving the scheme's security results in a security reduction to some hard underlying problems. Intuitively, this means that breaking the security of the cryptographic scheme with respect to the model is as difficult as solving the underlying hard problem. Because it is unknown how to solve the underlying hard problems efficiently, the security of the cryptographic scheme follows.

3.1.2 Definition of the Interfaces

The formalization of the interfaces usually starts with an intuition of a certain real-world application. As an example, consider the case shown in Figure 1 in which Alice and Bob wish to exchange encrypted messages. To realize this application,

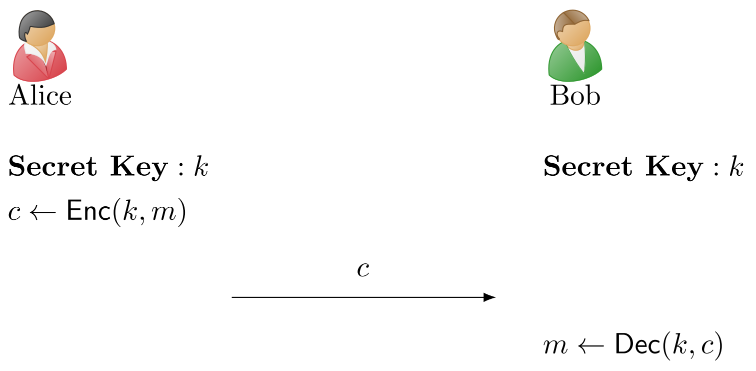

Alice
Bob
Secret Key : k
Secret Key : k
$c \leftarrow \mathsf{Enc}(k, m)$
c
$m \leftarrow \mathsf{Dec}(k, c)$

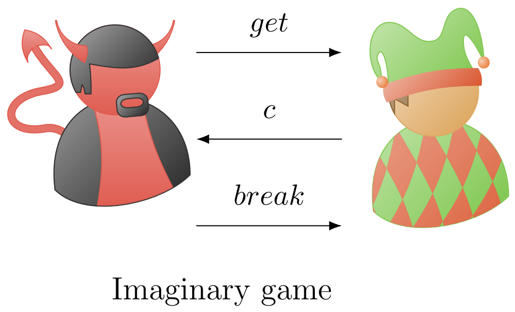
Figure 1: Intuitive description of Alice sending Bob an encrypted message.

Alice and Bob need an algorithm to encrypt and decrypt messages. For each of these operations, we define the algorithm Enc for encryption and Dec for decryption; the inputs of each algorithm are shown in Figure 1. Since each algorithm requires some private key as input, we must describe an algorithm Gen to compute this key. All algorithms together $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ describe the interfaces of an encryption scheme. Once the intuitive description is finished, we need to specify the functionality, i.e., each algorithm's precise input and output behavior. Finally, we describe the cryptographic scheme's expected (honest) behavior, called correctness or completeness. In our running example, correctness implies that the (honest) decryption of an honestly generated ciphertext yields the plaintext.

3.1.3 Cryptographic Games

The basic idea of a game-based security notion is to describe the security properties as a game between an adversary and a challenger. The adversary's goal is to attack a cryptographic scheme, and to do so, he expects to interact with the scheme to some degree. For example, suppose that the adversary is attacking an encryption scheme, and let's assume that the attacker needs some ciphertext to carry out the attack. Where is the ciphertext coming from? This is where the challenge comes into play. The challenger generates the entire "world" for the adversary and provides all interfaces that the adversary expects. Eventually, the adversary finishes the attack and outputs some value. The challenger then evaluates this output and tells if it is a valid attack (in this formal model) or not.

We explain this intuitive idea with the example shown in Figure 2. The left-hand side shows the adversary attacking some cryptographic scheme Π. In our running example, the adversary $\mathcal{A}$ attacks some encryption scheme. According to



```
Exp_{Π,A}(λ)
1 : state ← Setup(1^λ)
2 : (get, st) ← A(1^λ)
3 : c ← Compute(state, get)
4 : break ← A(st, c)
5 : b ← Eval(state, c, break)
6 : return b
```

Figure 2: Visualization of a cryptographic game and the corresponding formalization.

our imaginary security notion, the only possibility for the adversary to attack the scheme is by requesting a single ciphertext c. To do so, the adversary sends the message *get* to the challenger, who responds with some ciphertext c. Eventually, the attacker sends some value *break* to the challenger and stops. The challenger then analyzes this value and outputs a decision whether *break* is a valid attack or not.

This intuitive interaction/game is formalized in a *cryptographic game* shown on the right-hand side. Formally, this game is a random variable that is called Exp, and it binds both the cryptographic scheme Π and the adversary $\mathcal{A}$ to it. The game consists of several steps executed from Lines 1 to 6. In the first step, the game generates some private state variable called state. This step might correspond to creating some system parameters and possibly some (private) cryptographic keys that may not be shared with the adversary. In the second step, the adversary is initialized with the security parameter 1^λ (see Section 3.2.1 for a description and explanation of this parameter) as input and he outputs the message *get* together with some private state st. This line corresponds to the first message in the intuitive execution. The private state reflects the knowledge of the attacker. Afterward, in Line 3, the challenger computes the value c using the private state state and possibly the message *get*. Subsequently, the attacker is executed on its private state st and on c, and he returns some string *break*. This string is completely arbitrary at this point. Depending on the security notion, it may be some message, some bit, or a signature. In Line 5, the challenger checks if *break* is a valid attack. In this case, this check involves some algorithm Eval that takes as input the state information state and the values c and *break*. The output of this algorithm is a bit b, with the obvious meaning.

The last step towards formalizing a game-based security notion is the definition of success, which might look as follows:

The cryptographic scheme Π is *secure*, if for all efficient adversaries $\mathcal{A}$ the following holds:

$$\Pr\left[\mathsf{Exp}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \leq 1/2 + \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\lambda)$ is a negligible function in the security parameter λ and where the probability is taken over the random choices of $\mathcal{A}$ and coin tosses of Π.

The formalization says that the adversary's success probability should be very small. However, there seems to be a trivial attack in this example by outputting a random bit. Thus, simply guessing, the attacker wins this game with a probability of at least 1/2. The observation here is that guessing the bit does not help the adversary either, i.e., by guessing, he does not learn any meaningful information. However, if the attacker wins the game by significantly winning the game with a higher probability than 1/2, he learns some non-trivial information. Therefore, this definition says that the adversary wins the game if he can break the security *non-negligibly* (cf. Section 3.2.1) bigger than 1/2, which essentially means better than guessing.

3.1.4 Reduction Proofs

We require the cryptographic schemes used in practical protocols to be *provably secure*. To show that a cryptographic construction Π is computationally secure, we relate its security to one of the (unproven) cryptographic hardness assumptions. This technique is denoted as *reduction* and works as follows. First, the construction we wish to prove realizes the interfaces and usually builds upon simple (unproven) assumptions. Some theorem will be of the form "Assume that some problem X is hard, then the construction Π is secure with respect to definition Y". The first step of this proof technique is to assume towards contradiction that our construction Π is *insecure* with respect to Definition Y. If this is the case, an efficient adversary $\mathcal{A}$ exists against the scheme Π *by definition*. Note that the definition tells us what the adversary is expecting and the conditions under which he succeeds. Then we show how to turn the adversary $\mathcal{A}$ into an algorithm $\mathcal{B}$ against the underlying problem X. By doing so, we can argue the security of our scheme as follows. If we assume that the problem X underlying the hardness assumption is infeasible, the algorithm $\mathcal{B}$ can not exist. This implies that the attacker $\mathcal{A}$ against the scheme Π (from which $\mathcal{B}$ derived his solution for the problem X) does not exist. Therefore, the construction Π is secure. In particular, a reduction proof proceeds via the following steps:

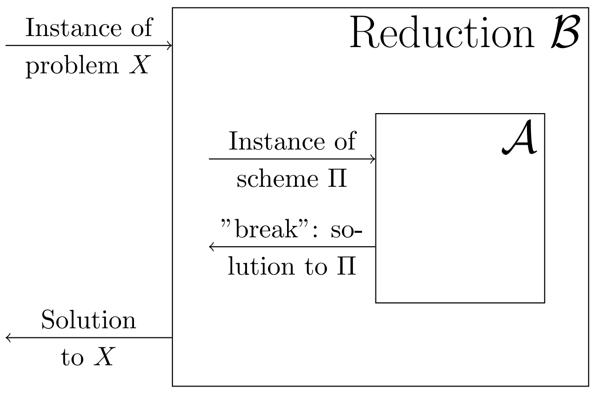


Figure 3: Proof by Reduction [1].

1. Fix some efficient (PPT) adversary $\mathcal{A}$ attacking Π. Denote the adversary's success probability by $\epsilon(\lambda)$.

2. Construct from $\mathcal{A}$ an efficient algorithm $\mathcal{B}$ for the problem X (called "reduction"). $\mathcal{B}$ uses $\mathcal{A}$ as a subroutine but does not know anything about how $\mathcal{A}$ works. The only thing $\mathcal{B}$ knows is that $\mathcal{A}$ expects to attack the cryptographic construction Π. So, given some input instance x of the problem X, algorithm $\mathcal{B}$ will simulate for $\mathcal{A}$ an instance of Π such that

   a) As far as $\mathcal{A}$ can tell, he interacts with Π. The view of $\mathcal{A}$ when he is used as a subroutine of $\mathcal{B}$ should be distributed identically to (or at least close to) the view of $\mathcal{A}$ when he interacts with Π.

   b) If $\mathcal{A}$ succeeds in "breaking" the instance of Π that $\mathcal{B}$ is simulating, $\mathcal{B}$ can derive a solution to the instance x of the problem X with at least inverse polynomial probability $1/p(\lambda)$.

3. Taking a) and b) together, we find that $\mathcal{B}$ solves the problem X with probability $\geq \epsilon(\lambda)/p(\lambda)$. If $\epsilon(\lambda)$ is non-negligible and $\mathcal{A}$ is efficient, $\mathcal{B}$ is an efficient algorithm that solves the problem X with non-negligible probability. This contradicts our initial assumption.

4. Given our assumption regarding problem X, we conclude that no efficient adversary $\mathcal{A}$ can succeed in breaking Π with non-negligible probability. Stated differently, Π is computationally secure.

The above proof is illustrated by Figure 3.

3.2 Hardness Assumptions and Mathematical Foundations

Most cryptographic schemes are not unconditionally secure. Instead, the security of the schemes relies on unproven hardness assumptions that are widely believed to hold. The security of these hardness assumptions is analyzed independently. In many cases, cryptoanalysis did not make significant progress, sometimes over hundreds of years, as in the case of the famous factoring problem, where the attacker receives as input the product $N = pq$ of two (large) primes p and q, and its task is to find these primes.

Other relaxations are the restriction of the running time of the adversary and its success probability. Modern cryptographic schemes assume the running time of the adversary is bounded by a polynomial that might be arbitrarily large. For most practical applications, this assumption is perfectly fine considering a running time of, e.g., hundred years exceeds the lifetime of most humans. The use of cryptographic keys leads to the fact that there is a small residual risk with which the attacker can guess this key. For example, if the key has 128 bits, then the attacker might guess this key with probability 2^{128}, which is a number with 39 decimal digits. This number can also be approximated with 3.4×10^{38}. In other words, the probability that the attacker guesses that key is $\frac{1}{3.4 \times 10^{38}}$. To better grasp this number, we can use the estimated number of atoms in the universe as a comparison, which is estimated to range from between 10^{78} to 10^{82} atoms. In many cases, this success probability depends on the computational power of the adversary, which doubles according to Moore's law every two years. This increase in computational power is expressed with the security parameter and covered by negligible functions, which we introduce in Section 3.2.1.

3.2.1 Security Parameter and Negligible Functions

A cryptographic security parameter 1^λ is a value used to determine the strength and security of a cryptographic algorithm or protocol. The security parameter is typically encoded in unary. Its value determines various aspects of the cryptographic scheme, such as the length of keys, the number of rounds in a cipher, or the size of a hash function. Unary encoding is a technique used in complexity theory and cryptography to ensure that certain algorithms and protocols run in polynomial time. By encoding a security parameter in unary, the length of the input to the algorithm or protocol is proportional to the parameter's value. This ensures that any algorithms or protocols that use the parameter will have a polynomial running time in the parameter value, since the input length is polynomial in the parameter value.

For example, if the security parameter is λ, then encoding it in unary results in a string of λ ones ("111...1"). Any algorithm or protocol that uses the security parameter would take a time polynomial in λ to process this input. This helps to ensure that the algorithm or protocol is efficient and secure, as it limits the potential for attacks based on exponential or superpolynomial running times.

Modern cryptography can be broken with some small probability. To formalize what "small" means, we introduce the notion of negligible functions. These are functions that are smaller than the inverse of any polynomial. More formally:

Definition 1 (Negligible Function)**.** A function f is *negligible* if for every polynomial $p(\cdot)$ there exists N such that for all integers $\lambda > N$ it holds that

$$|f(\lambda)| < \frac{1}{|p(\lambda)|}.$$

To give an intuition for this definition, we propose the following examples:

$2^{-\lambda}$ This is a negligible function since it is exponential and exponential functions rise faster than polynomials.

$\lambda \in \mathbb{N}$ For $\lambda \neq 0$ this is not a negligible function as $\lim_{n\to\infty} |p(\lambda)| = \infty > \lambda$. For $\lambda = 0$ this is negligible with the same argument.

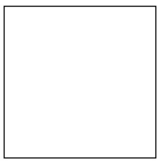
$\frac{1}{p^2}$ This is not a negligible function, since the polynomial $\frac{1}{p^3}$, it holds that $\lim_{n\to\infty} \frac{p^2(\lambda)}{p^3(\lambda)} = \frac{1}{p(\lambda)} = 0$. So there exists a N such that $\frac{1}{p(\lambda)^2} \geq \frac{1}{|p(\lambda)|^3}$ for $\lambda > N$.

Proposition 1. Let $\mathsf{negl}(\lambda)_1$ and $\mathsf{negl}(\lambda)_2$ be negligible functions. Then:

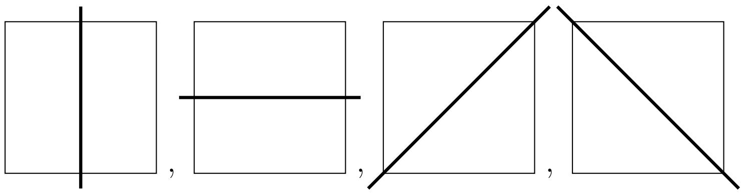
- $\mathsf{negl}(\lambda)_3 = \mathsf{negl}(\lambda)_1 + \mathsf{negl}(\lambda)_2$ is negligible.
- For any positive polynomial p, the function $negl_4 = p(\lambda) \cdot \mathsf{negl}(\lambda)_1$ is negligible.

3.2.2 Mathematical Foundations

A group is a mathematical structure consisting of a set $\mathbb{G}$ and an operation $\star : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$. The basic fact here is that, if we apply the group operation to two group elements, we will obtain another group element. Furthermore, the group must contain a neutral element id and, for each group element a an inverse element a^{-1} with $a \star a^{-1} = id$. An illustrating example is that of the symmetry group of a geometric object, e.g., a square.



The square has eight symmetries: Four rotational symmetries and four reflection symmetries.
With respect to rotation, we can rotate the square by 90, 180, 270 or 360 degree without changing its appearance. With respect to reflection, we can reflect the square on each of the four axes depicted below.



We now give a formal definition of the term group.

Definition 2 (Group)**.** A *group* $(\mathbb{G}, \star)$ consists of a set $\mathbb{G}$ and an operation $\star : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ fulfilling the following properties

- Closure: $\forall\ a, b \in \mathbb{G}$ we have $a \star b \in \mathbb{G}$
- Associativity: $a \star (b \star c) = (a \star b) \star c\ \ \forall\ a, b, c \in \mathbb{G}$
- neutral element: $\exists\ [0] \in \mathbb{G}$ such that $[0] \star a = a \star [0] = a\ \ \forall\ a \in \mathbb{G}$
- inverse elements: $\forall\ a \in \mathbb{G}\ \exists\ a^{-1} \in \mathbb{G}$ such that $a \star a^{-1} = a^{-1} \star a = 1$.

It can be shown that the neutral element 1 of the group $\mathbb{G}$ is unique. Furthermore, for every $a \in \mathbb{G}$, the inverse element a^{-1} is uniquely determined.

3.2.3 The Discrete Logarithm Assumption

The first computational assumption used within this analysis is the discrete logarithm assumption. Intuitively, this problem says that given an element g^x, it is computationally difficult to compute the exponent x (cf. left part of Figure 4).

```
$\mathsf{DLog}_{\mathcal{A},\mathsf{GGen}}(\lambda)$
1 : $(\mathbb{G}, [1], q) \leftarrow \mathsf{GGen}(1^\lambda)$
2 : $x \leftarrow\!\!\$\ \mathbb{Z}_q$
3 : $x' \leftarrow \mathcal{A}(\mathbb{G}, q, [1], [x])$
4 : $\mathbf{return}\ ([x] = [x'])$

$\mathsf{DDH}_{\mathcal{A},\mathsf{GGen}}(\lambda)$
1 : $(\mathbb{G}, [1], q) \leftarrow \mathsf{GGen}(1^\lambda)$
2 : $x, y, c_0 \leftarrow\!\!\$\ \mathbb{Z}_q$
3 : $c_1 := xy$
4 : $b \leftarrow\!\!\$\ \{0, 1\}$
5 : $b' \leftarrow \mathcal{A}(\mathbb{G}, q, [1], [x], [y], [c_b])$
6 : $\mathbf{return}\ (b = b')$
```

Figure 4: Cryptographic games for the discrete logarithm, and decisional Diffie-Hellman assumption.

Notation	*Description*
$\mathbb{G}$	Cyclic groups of order q
$[1]$	A generator of $\mathbb{G}$
$[0]$	Identity element of $\mathbb{G}$
$[a], a \in \mathbb{Z}_q$	Group element in $\mathbb{G}$ with discrete logarithm $a \in \mathbb{Z}_q$ with respect to $[1]$
$[A] + [B] = [A + B]$	Entrywise group operation
$c \cdot [A] := [A] \cdot c := [c \cdot A]$	Exponentiation of each entry of $[A]$ by c

Table 1: Summary of the notation for group operations

In general, the difficulty of computational problems always depends on an underlying structure, in most cases, a group. In fact, for the discrete logarithm problem, there are some groups known in which the problem is computationally easy, and in other groups, the problem is believed to be hard. Therefore, we first define the notion of a group generation algorithm, and all hardness assumptions will always hold for some group that is generated by that algorithm. Table 1 summarizes the notion we use in this analysis.

Definition 3 (Group generation algorithm)**.** Let $(\mathbb{G}, [1], q) \leftarrow \mathsf{GGen}(1^\lambda)$ be an efficient generic algorithm which, on input of a security parameter 1^λ, outputs a cyclic group $\mathbb{G}$ of order q (q is a λ-bit integer) with generator $[1] \in \mathbb{G}$.

With the definition of the group generation algorithm, we introduce the discrete logarithm problem.

Definition 4 (Discrete logarithm problem)**.** The *discrete logarithm problem* with regard to GGen is hard if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\Pr\left[\mathsf{DLog}_{\mathcal{A},\mathsf{GGen}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda),$$

where the game $\mathsf{DLog}_{\mathcal{A},\mathsf{GGen}}$ is defined in Figure 4.

Examples The discrete logarithm problem can be defined in any cyclic group $\mathbb{G}$. However, it is not equally hard in all groups. In this section, we give some examples of cyclic groups which are suited or not suited for discrete logarithm cryptography.

- In the cyclic group $(\mathbb{Z}_p, +)$ for p being prime, the discrete logarithm problem is easy. Given a generator $g \in \mathbb{Z}_p$ and another group element $h = a \cdot g$, we can easily find the secret value a using the extended euclidean algorithm. $h = a \cdot g \mod p$ implies that $h = a \cdot g + k \cdot p$ holds in $\mathbb{Z}$. Since $\gcd(a, p) = 1$ holds for any $a \in \mathbb{Z}_p$, we can use the extended euclidean algorithm to find integers a' and k' such that $a' \cdot g + k' \cdot p = 1$. Therefore we have $ha' \cdot g + hk' \cdot p = h$, which implies that $ha' \mod p$ is a solution to the given discrete logarithm problem.
- In the cyclic group $(\mathbb{Z}_p^\star, \cdot)$, the discrete logarithm problem is believed to be hard. However, the group $(\mathbb{Z}_p^\star, \cdot)$ is not a group of prime order. This enables the use of special algorithms like the Pohlig-Hellman algorithm, which make use of the factorization of the group order to speed up attacks against the

discrete logarithm problem. Therefore we often use subgroups of $\mathbb{Z}_p^\star$ of prime order in cryptographic applications.

- In a cyclic subgroup of the group of points on an elliptic curve $E(\mathbb{Z}_p)$, the discrete logarithm problem is believed to be hard. Furthermore, some algorithms for solving discrete logarithms in $\mathbb{Z}_p^\star$ (e.g., the index-calculus techniques) do not work in this case. This allows us to choose smaller parameters for the cryptographic schemes working in $E(\mathbb{Z}_p)$.

3.2.4 The Diffie-Hellman Assumption

For some proofs, the discrete logarithm problem is insufficient, and stronger computational assumptions are required. Therefore, a widely used assumption is the decisional Diffie-Hellman assumption we introduce in the following. Intuitively, the decisional Diffie-Hellman problem says that given the tuple g^x, g^y, g^z, no efficient algorithm can tell if $z = xy$ or if z is a random element in $\mathbb{Z}_p$.

Definition 5 (Decisional Diffie-Hellman (DDH) problem)**.** The Decisional Diffie-Hellman problem with regard to GGen is said to be hard if, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\Pr[\mathsf{DDH}_{\mathcal{A},\mathsf{GGen}}(\lambda) = 1] - \frac{1}{2} \leq \mathsf{negl}(\lambda),$$

where the game $\mathsf{DDH}_{\mathcal{A},\mathsf{GGen}}$ is defined in Figure 4.

The DDH problem says that there exists an algorithm GGen relative to which the DDH problem is hard. Obviously, if the discrete logarithm problem with regard to GGen is easy, the same is true for DDH.

3.2.5 The Gap Diffie-Hellman Assumption

Intuitively, the Gap Diffie-Hellman assumption [5] says that given a triple (g, g^a, g^b), no efficient algorithm can find the element $C = g^{ab}$ with the help of a Decisional Diffie-Hellman Oracle.

Definition 6 (Gap Diffie-Hellman (GDH) problem)**.** The Gap Diffie-Hellman problem with regard to GGen [5] is said to be hard if, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\Pr[\mathsf{GDH}_{\mathcal{A},\mathsf{GGen}}(\lambda) = 1] - \frac{1}{2} \leq \mathsf{negl}(\lambda),$$

where the game $\mathsf{GDH}_{\mathcal{A},\mathsf{GGen}}$ is defined in Figure 5.

Obviously, if the discrete logarithm problem with regard to GGen is easy, the same is true for GDH.

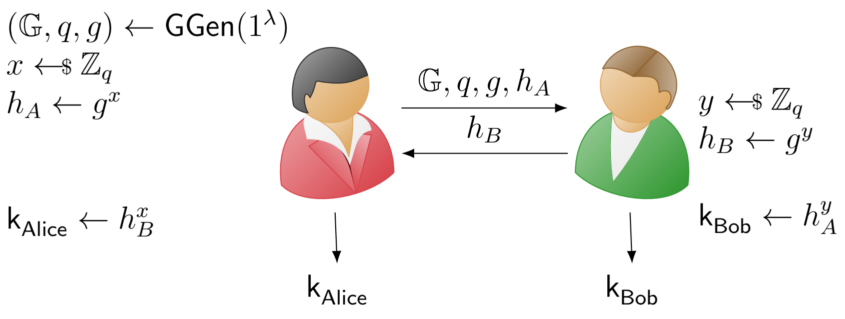
```
$\mathsf{GDH}_{\mathcal{A},\mathsf{GGen}}(\lambda)$
1 :  $(\mathbb{G}, [1], q) \leftarrow \mathsf{GGen}(1^\lambda)$
2 :  $x, y \leftarrow\!\!\$\ \mathbb{Z}_q$
3 :  $[z] \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{DDH}}}(\mathbb{G}, [1], [x], [y])$
4 :  return $([z] = [xy])$

$\mathcal{O}_{\mathsf{DDH}}([c], [x], [y])$
1 :  return $[xy] = [c]$
```

Figure 5: Cryptographic game for the Gap Diffie-Hellman assumption.

3.3 The Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange was invented in 1976; it is used in TLS 1.3 as the default key exchange protocol and serves as the fundamental building block for this protocol. The security is based on the DDH assumption we introduced in Section 3.2.3. In the Diffie-Hellman key exchange protocol, the two parties Alice and Bob exchange two values $h_{\mathsf{Alice}} = g^x$ and $h_{\mathsf{Bob}} = g^y$ for random exponents $x, y \in \mathbb{Z}_q$. Having access to h_{Alice} and y, Bob can now compute g^{xy} locally. The same key can be computed by Alice knowing h_{Bob} and x. Each eavesdropping party that sees only the transcript $\mathsf{Trans} = (\mathbb{G}, g, q, h_{\mathsf{Alice}}, h_{\mathsf{Bob}})$ and can tell this resulting key g^{xy} apart from a randomly sampled key can also break the CDH assumption on this instance. Therefore, DH key exchange is secure as long as CDH holds. We give a formal description of the protocol in Figure 6.



$(\mathbb{G}, q, g) \leftarrow \mathsf{GGen}(1^\lambda)$
$x \leftarrow\!\!\$\ \mathbb{Z}_q$
$h_A \leftarrow g^x$
$\mathbb{G}, q, g, h_A$
h_B
$y \leftarrow\!\!\$\ \mathbb{Z}_q$
$h_B \leftarrow g^y$
$\mathsf{k}_{\mathsf{Alice}} \leftarrow h_B^x$
$\mathsf{k}_{\mathsf{Bob}} \leftarrow h_A^y$
$\mathsf{k}_{\mathsf{Alice}}$
$\mathsf{k}_{\mathsf{Bob}}$

Figure 6: The Diffie-Hellman Key Exchange Protocol

Theorem 1. *If the decisional Diffie-Hellman problem is hard relative to* GGen*, then the Diffie-Hellman key exchange protocol* Π *is secure in the presence of an eavesdropper (w.r.t. the modified experiment* $\mathsf{KE}'^{eav}_{\mathcal{A},\Pi}(\lambda)$*).*

The Diffie-Hellman key exchange protocol is *correct*, if $\mathsf{k}_{\mathsf{Alice}} = \mathsf{k}_{\mathsf{Bob}}$:

$$\mathsf{k}_{\mathsf{Alice}} = h_B^x = (g^y)^x = g^{x \cdot y}$$

$$\mathsf{k}_{\mathsf{Bob}} = h_A^y = (g^x)^y = g^{x \cdot y}$$

3.4 The Random Oracle Model

Cryptographic schemes often require a hash function that is both collision resistant and truly random. However, a single function cannot have both properties simultaneously. Furthermore, since the key of a hash function is usually public, a pseudorandom function is not a viable solution. To address this challenge, three possible solutions have been proposed: relying on schemes that can be proven secure without the idealized primitive, employing schemes that require the idealized primitive without formal proof, or adopting a "middle ground" approach through the Random Oracle (RO) model. The RO model has a long history in cryptography and was first formalized by Bellare and Rogaway [10]. The RO model assumes the existence of a public oracle H that implements a truly random function. It is not hard to see that such an oracle cannot exist in reality, but this model provides a formal methodology to design and validate the security of cryptographic schemes following a typical two-step approach [1]:

1. The first step is the design of the scheme and providing a proof of security in the RO model. The construction might be based on "standard" cryptographic assumptions.

2. To use the scheme in the real world, each party uses a *real-world* hash function H' (such as e.g., BLAKE2b) and we adjust the scheme appropriately. Whenever the scheme asks to evaluate the RO on a value x, the function $H'(x)$ is computed locally.

The intention is that the hash function sufficiently emulates the Random Oracle (RO) to the extent that the security proof remains applicable in the standard model. To a certain degree, this intention is satisfied by the use of BLAKE2b as we show in Section 8.

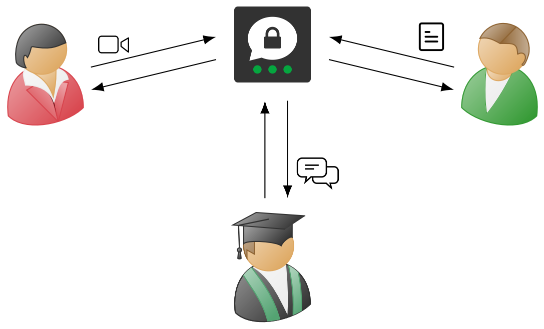
4 Formalization of the Functionality

The formalization of the interfaces may change during the formalization of the protocol.

Given the high-level description from the Ibex protocol, this section provides a formalization of the functionality of Ibex and its main core cryptographic components, such as the protocols for key exchange and secure channels. This formalization serves as the basis of the security analysis of Section 7.

4.1 Functionality of Ibex

We recall the manifold use cases of Threema, where multiple users want to exchange data securely in an asynchronous setting without the need of sharing private information in the first place.

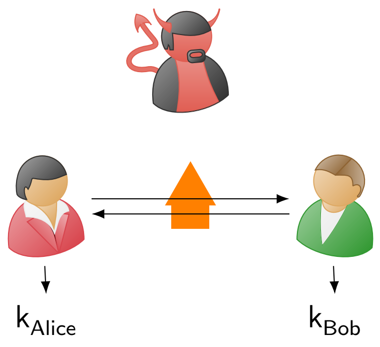


To achieve the desired functionality, we split the communication into two parts: In the first part, two users that do not share any private information want to authenticate each other and agree on a shared secret. In the second part, the two users want to use the shared secret they agreed on to exchange the data securely. In Ibex, the first part is realized using a key exchange protocol called 4DH, and the second part is realized using a secure channel protocol.

In our formalization, we split the key exchange and channel components as follows: whenever a new symmetric key for authenticated encryption is computed, this key computation is part of the *key exchange*, and only the payload encryption is part of the *channel*. As a result, the key exchange component is actually a *continuous key exchange*. We now formalize these two protocols.

4.2 Continuous Key Exchange Protocols

On an intuitive level, a continuous key exchange protocol allows two parties to agree on a series of shared keys that are hidden from any eavesdropping adversary. We build a security experiment for continuous key exchange protocols according to this intuition as follows: In the first step, two parties, Alice and Bob, run the two-party key exchange protocol and output the transcript Trans of this protocol run, as well as resulting keys k. The transcript is all the information that Alice and Bob exchange during the execution of the protocol. Then we sample a random bit b. If $b = 0$, we set each challenge key k' to be an original output of the computation of Alice and Bob. If the bit is 1, we sample the challenge keys k' to be random. Finally, the adversary gets access to the challenge keys k' and has to decide if it is a randomly sampled one or the output of the transcript Trans. If the continuous key exchange protocol is correct, then the series of resulting keys $\mathsf{k}_{\mathsf{Alice}}$ of Alice and $\mathsf{k}_{\mathsf{Bob}}$ of Bob are the same.



The description given here suffices to get an intuition of how continuous key exchange protocols work. The protocol Ibex has a more complex application and, therefore, we introduce the following more complex interface.

Definition 7 (Continuous Key Exchange Protocol)**.** A continuous key exchange protocol $\Pi_{\mathsf{KE}} = (\mathsf{Gen}_{\mathsf{KE}}, \mathsf{send}_{\mathsf{KE}}, \mathsf{recv}_{\mathsf{KE}})$ is defined as follows

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}_{\mathsf{KE}}(\lambda)$**:** The generation algorithm $\mathsf{Gen}_{\mathsf{KE}}$ is a PPT algorithm that on input of the security parameter λ outputs a long-term key pair $(\mathsf{sk}, \mathsf{pk})$ consisting of a secret key sk and the corresponding public key pk.

$(\mathsf{st}'_S, \mathsf{k}, c) \leftarrow \mathsf{send}_{\mathsf{KE}}(\mathsf{st}_S, \mathsf{sk}_S, \mathsf{pk}_R)$**:** The sending algorithm $\mathsf{send}_{\mathsf{KE}}$ is a PPT algorithm that on input of a long-term secret key sk_S and a state of ephemeral secrets st_S of the sender as well as a long-term public key pk_R of the receiver outputs an updated state st'_S of the sender, a shared symmetric key k, and a ciphertext c.

$(\mathsf{st}'_R, \mathsf{k}, \mathsf{pk}_S) \leftarrow \mathsf{recv}_{\mathsf{KE}}(\mathsf{st}_R, \mathsf{sk}_R, c)$**:** The receiving algorithm $\mathsf{recv}_{\mathsf{KE}}$ is a DPT algorithm that on input of a long-term secret key sk_R and a state of ephemeral secrets st_R of the receiver as well as a ciphertext c outputs an updated state st'_R of the receiver, a shared symmetric key k, and a long-term public key pk_S of the corresponding sender.

4.3 Channel Protocols

A vital function of modern communication infrastructure is to ensure the protection of data during transmission in various connections each day. At a fundamental level, a secure channel is a communication channel that guarantees the protection of data transmitted over it from unauthorized access, tampering, and interception by malicious third parties. To establish secure channels, cryptographic protocols that utilize encryption and authentication techniques are commonly used to protect the data in transit. When a party Alice wants to send a message over a secure channel to Bob, this works the following way on a high level: Alice holds a state $\mathsf{state}_{\mathsf{Alice}}$ and a message m, and Bob holds his state $\mathsf{state}_{\mathsf{Bob}}$. The sender, Alice, encrypts the tuple $(\mathsf{state}_{\mathsf{Alice}}, m)$ and obtains thereby the values $(\mathsf{state}'_{\mathsf{Alice}}, c)$, where $\mathsf{state}'_{\mathsf{Alice}}$ is her updated state, and c is the corresponding ciphertext to the message m. When Bob receives a ciphertext c, he uses the tuple $(\mathsf{state}_{\mathsf{Bob}}, c)$ to decrypt the ciphertext c to a message m'. This decryption leads to a newly updated state $\mathsf{state}'_{\mathsf{Bob}}$.

Definition 8 (Secure Channel)**.** A secure channel protocol $\Pi_{\mathsf{SC}} = (\mathsf{Gen}_{\mathsf{SC}}, \mathsf{send}_{\mathsf{SC}}, \mathsf{recv}_{\mathsf{SC}})$ is defined as follows:

$(\mathsf{state}_{\mathsf{Alice}}, \mathsf{state}_{\mathsf{Bob}}) \leftarrow \mathsf{Gen}_{\mathsf{SC}}(\lambda)$**:** The state generation algorithm is a PPT algorithm that on input of the security parameter λ outputs the states $(\mathsf{state}_{\mathsf{Alice}}, \mathsf{state}_{\mathsf{Bob}})$ of Alice, resp. Bob.

$(\mathsf{state}', c) \leftarrow \mathsf{send}_{\mathsf{SC}}(\mathsf{state}, m)$**:** The sending algorithm $\mathsf{send}_{\mathsf{SC}}$ is a PPT algorithm that on input of a state state and a message m outputs an updated state state' and a ciphertext c.

$(\mathsf{state}', m, \mathsf{ctr}) \leftarrow \mathsf{recv}_{\mathsf{SC}}(\mathsf{state}, c)$**:** The receiving algorithm $\mathsf{recv}_{\mathsf{SC}}$ is a DPT algorithm that on input of a state state and a ciphertext c outputs an updated state state' and a message m.

5 Formalization of the Security Properties

The core security properties that a messaging system should provide are *confidentiality*, *integrity*, and *authenticity*. To analyze the security of such a system, it makes sense to define and analyze the security properties of its underlying components. This modular approach simplifies and clarifies the analysis.

5.1 Security Properties in General

Confidentiality preserves authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information.

Integrity guards against improper information modification or destruction. We distinguish between data integrity and system integrity.

Data Integrity The property that data has not been altered in an unauthorized manner. Data integrity covers data in storage, during processing, and while in transit.

System Integrity The quality that a system has when it performs its intended function in an unimpaired manner, free from unauthorized manipulation of the system, whether intentional or accidental.

Authenticity guarantees that the origin of an operation or the source of data cannot be altered.

Forward Security When secret key material is compromised, all information that was protected by that key material *before* the compromise remains secure; information that is protected by that key material *after* the compromise may become insecure. Forward security is an orthogonal dimension to the prior security guarantees. For example, confidentiality is extended by forward security—and, thereby, usually called *Forward Secrecy*—when all past information remains private and only future information is revealed due to the compromise of the used key material. The naming in the cryptographic literature is remarkably inconsistent by introducing several variants like ‘perfect’, ‘strong’, ‘weak’, etc. forward security, where the actual meaning is often ambiguous and sometimes even counterintuitive[1]. For that reason, we only use the term ‘forward security’ in this document.

[1]See a discussion about the naming and distinction of ‘forward security’, ‘backward security’, ‘future security’, and ‘post-compromise security’ in [20]

5.2 Security Properties of Continuous Key Exchange

We capture all security properties relevant to our analysis of the continuous key exchange protocol of Threema in one definition. For this, we consider the composition of Threema's 4DH protocol as well as the subsequent key update mechanism, as one building block. Compactly, our definition requires that the keys that this building block computes are indistinguishable from random keys even if future mid-term and long-term key material is compromised at some point. Based on this definition, we capture: *Confidentiality* of keys, *Implicit Authentication* of keys and users, *Replay Attack Resilience*, and *Forward Security* of keys. The authentication property is reached implicitly because the protocol does not abort upon an active attack by a network adversary. Instead, the keys computed by the protocol participants are independent of such an attack.

5.3 Game-based Definition for Continuous Key Exchange

For our formal definition, we provide a game that captures the above intuition. In this game, an adversary can generate new parties, let parties establish keys with other parties, let parties process manipulated ciphertexts—to model active network attacks—, corrupt mid-term and long-term key material, and ask for key challenges. As sketched above, the latter means that the adversary either obtains a real key established between two parties or a random key. Based on this challenge, the adversary has to guess which of both it obtained. Our upcoming proof will show that guessing correctly is practically infeasible.

The sketched game is defined by specifying the following oracles that the adversary can query to give the adversary full control of the execution environment:

Gen: Generates a new party in the environment. In practice, this models that a new user is registered.

Send: Lets a party send to another party and, thereby, establish a new symmetric key. In practice, this key is then used to encrypt a plaintext message to the receiver. Due to our modular approach, the encryption step is captured in the subsequent channel part.

$\mathsf{Challenge}$: Lets a party send to another party and, thereby, establish a new symmetric key. In contrast to oracle Send, oracle $\mathsf{Challenge}$ either outputs the real established key or a random key.

Recv: Lets a party receive a ciphertext. This ciphertext can be the output of oracles Send or $\mathsf{Challenge}$, or this ciphertext was crafted by the adversary to model active network attacks. If algorithm $\mathsf{Recv}_{\mathsf{KE}}$ accepts an input ciphertext with output pk_j but party pk_j never sent that ciphertext due to oracle queries $\mathsf{Send}(j, i)$ or $\mathsf{Challenge}(j, i)$ or this ciphertext was received by an earlier query to oracle $\mathsf{Challenge}(i, \cdot)$ already, then that ciphertext as well as all subsequent ciphertexts in that session are considered *out-of-sync* (i.e., the opposite of *in-sync*).

$\mathsf{Corrupt}$: Compromises the long-term key as well as mid-term keys of a corrupted party. This models that the party could unintentionally store these keys on a publicly accessible memory or that real-world attackers may obtain physical access to victims' devices or plant viruses thereon.

```
**Game** $\mathsf{Exp}^{b}_{\mathsf{KE},\mathcal{A}}(\lambda)$
00 $n \leftarrow 0$
01 $b' \leftarrow \mathcal{A}(1^{\lambda})$
02 Return $b'$

**Oracle** $\mathsf{Gen}(\lambda)$
03 $n \leftarrow n + 1$
04 $(\mathsf{sk}_n, \mathsf{pk}_n) \leftarrow \mathsf{Gen}_{\mathsf{KE}}(1^{\lambda})$
05 Return $\mathsf{pk}_n$

**Oracle** $\mathsf{Send}(i, j)$
06 $(\mathsf{st}_i, \mathsf{k}, c) \leftarrow \mathsf{send}_{\mathsf{KE}}(\mathsf{st}_i, \mathsf{sk}_i, \mathsf{pk}_j)$
07 Return $(\mathsf{k}, c)$

**Oracle** $\mathsf{Challenge}(i, j)$
08 $(\mathsf{st}_i, \mathsf{k}_0, c) \leftarrow \mathsf{send}_{\mathsf{KE}}(\mathsf{st}_i, \mathsf{sk}_i, \mathsf{pk}_j)$
09 $\mathsf{k}_1 \leftarrow \{0,1\}^{\lambda}$
10 Return $(\mathsf{k}_b, c)$

**Oracle** $\mathsf{Recv}(i, c)$
11 $(\mathsf{st}_i, \mathsf{k}, \mathsf{pk}) \leftarrow \mathsf{recv}_{\mathsf{KE}}(\mathsf{st}_i, \mathsf{sk}_i, c)$
12 If *in-sync*: Return $\mathsf{pk}$
13 Return $(\mathsf{k}, \mathsf{pk})$

**Oracle** $\mathsf{Corrupt}(i)$
14 Return $(\mathsf{sk}_i, \mathsf{st}_i)$
```

Figure 7: Cryptographic game to define the security of continuous key exchange as instantiated by 4DH.

These oracles are formally specified in Figure 7. Before we can define what security means, we have to identify and exclude certain *trivial winning conditions*. Under these conditions, it is easy to win the game, even for efficient adversaries. However, these attacks are meaningless in practice:

1. After party i is corrupted, all keys established by subsequent queries to oracle $\mathsf{Challenge}(\cdot, i)$ are considered insecure.
   Forward security only requires that past keys remain secure but not that future keys will be secure again.

2. Similarly, after party i is corrupted, all keys established by earlier queries to oracle $\mathsf{Challenge}(\cdot, i)$ that were not received via corresponding queries to oracle $\mathsf{Recv}(i, \cdot)$ are considered insecure.
   By functionality, it is necessary that the corrupted secrets can compute keys that were established by the communication partner but not yet received.

3. Furthermore, after party i is corrupted, all keys established by queries to oracle $\mathsf{Challenge}(j, i)$ that were issued before partner j received a ciphertext from party i via corresponding queries to oracle $\mathsf{Recv}(j, \cdot)$ are considered insecure.
   The first half round-trip in a session only offers forward security from the session initiator to the session responder but not vice versa (cf. 2DH vs. 4DH).

4. Finally, after party i is corrupted, all keys established by subsequent queries to oracle $\mathsf{Challenge}(i, \cdot)$ are considered insecure. This attack is not trivial in general[2], but since Ibex uses symmetric secrets for deriving (future) challenge keys (in established) sessions, a corruption of party i reveals future keys established by i and to i.

[2]By using asymmetric secrets, the state of i would not reveal keys established by i.

If any of the keys output by oracle Challenge is considered insecure, we let the adversary automatically lose the game. For clarity and compactness, we refrain from encoding these trivial winning conditions (and how they are prevented) in Figure 7.

Based on this, we define security by requiring that it should be hard for an adversary to correctly guess the random parameter bit b when terminating with return value b'. More concretely:

Definition 9 (Secure Continuous Key Exchange)**.** A continuous key exchange protocol KE is secure if the advantage

$$\mathsf{Adv}^{\mathsf{exp}}_{\mathsf{KE},\mathcal{A}}(\lambda) := \left|\Pr[\mathsf{Exp}^{0}_{\mathsf{KE},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{1}_{\mathsf{KE},\mathcal{A}}(\lambda) = 1]\right|$$

is negligible for all PPT adversaries $\mathcal{A}$.

5.4 Security Properties of Simple Channels

Since the continuous key exchange component already captures most of the desired security properties, the channel notion considered here can be relatively simple: We treat the channel as a basic authenticated encryption scheme. Such an authenticated encryption scheme has to provide three basic properties: *Confidentiality*, *Integrity*, and *Explicit Authentication*. Here, authentication is explicit because the recipient must reject manipulated ciphertexts.

5.5 Game-based Definition for Authenticated Encryption

We adopt the established game-based notion of confidentiality and ciphertext-integrity for authenticated encryption from [4]. The formal definitions can be found in Figure 8.

```
Game INT($\mathcal{A}$)
$\mathsf{k} \leftarrow \{0,1\}^\lambda$
$\mathrm{Q} \leftarrow \emptyset$
$\mathrm{N} \leftarrow \emptyset$
Invoke $\mathcal{A}$
Stop with 0

Oracle Encrypt($n, \mathsf{ad}, m$)
Require $n \notin \mathrm{N}$
$c \leftarrow \mathsf{Enc}(k, n, \mathsf{ad}, m)$
$\mathrm{Q} \leftarrow \mathrm{Q} \cup \{(n, \mathsf{ad}, c)\}$
$\mathrm{N} \leftarrow \mathrm{N} \cup \{n\}$
Return $c$

Oracle Decrypt($n, \mathsf{ad}, c$)
$m \leftarrow \mathsf{Dec}(k, n, \mathsf{ad}, c)$
If $m = \bot$: Return $\bot$
Reward $(n, \mathsf{ad}, c) \notin \mathrm{Q}$
Return $m$
```

```
Game IND$^b$($\mathcal{A}$)
$k \leftarrow \{0,1\}^\lambda$
$\mathrm{Q} \leftarrow \emptyset$
$\mathrm{N} \leftarrow \emptyset$
$b' \leftarrow \mathcal{A}$
Stop with $b'$

Oracle Encrypt($n, \mathsf{ad}, m^0, m^1$)
Require $n \notin \mathrm{N}$
Require $|m^0| = |m^1|$
$c \leftarrow \mathsf{Enc}(k, n, \mathsf{ad}, m^b)$
$\mathrm{Q} \leftarrow \mathrm{Q} \cup \{(n, \mathsf{ad}, c)\}$
$\mathrm{N} \leftarrow \mathrm{N} \cup \{n\}$
Return $c$

Oracle Decrypt($n, \mathsf{ad}, c$)
$m \leftarrow \mathsf{Dec}(k, n, \mathsf{ad}, c)$
If $m = \bot$: Return $\bot$
If $(n, \mathsf{ad}, c) \in \mathrm{Q}$:
    $m \leftarrow \diamond$
Return $m$
```

Figure 8: Cryptographic game to define the security of authenticated encryption.

6 Formalization of the Protocol

This section describes the cryptographic protocol called Ibex. This protocol's goal is to establish a secure channel between two users. In subsection 6.1 an intuitive description of the protocol is given. subsection 6.2 presents the corresponding formalization.

6.1 Intuitive Description

The goal of the Ibex protocol is the establishment of a forward secure channel between two users, Alice and Bob. Since Threema does not store the ephemeral secrets of its users on a centralized server, forward security can only be achieved after one communication round. Therefore, we assign each party a specific state to reflect the different states of the protocol and the corresponding security level that it achieves. The local state of Alice and Bob is of the form $\mathsf{state} \in \{\bot, \mathsf{2DH}_\mathsf{S}, \mathsf{2DH}_\mathsf{R}, \mathsf{4DH}\}$. Figure 9 provides a high-level overview of how the states transition into each other, and a more formal description is given below.

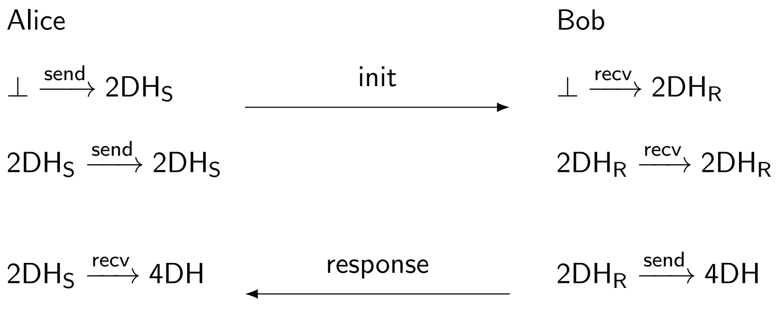

$$\begin{tikzcd}
\mathsf{Alice} & & \mathsf{Bob} \\
\bot \xrightarrow{\mathsf{send}} \mathsf{2DH}_\mathsf{S} \arrow[rr, "\mathsf{init}"] & & \bot \xrightarrow{\mathsf{recv}} \mathsf{2DH}_\mathsf{R} \\
\mathsf{2DH}_\mathsf{S} \xrightarrow{\mathsf{send}} \mathsf{2DH}_\mathsf{S} & & \mathsf{2DH}_\mathsf{R} \xrightarrow{\mathsf{recv}} \mathsf{2DH}_\mathsf{R} \\
\mathsf{2DH}_\mathsf{S} \xrightarrow{\mathsf{recv}} \mathsf{4DH} & & \mathsf{2DH}_\mathsf{R} \xrightarrow{\mathsf{send}} \mathsf{4DH} \arrow[ll, "\mathsf{response}"']
\end{tikzcd}$$

Figure 9: The possible local states in a channel initiated by Alice.

$\bot$ The initial state of both parties, when no communication happened between Alice and Bob so far, is $\bot$. Sending a message in the $\bot$ state leads to the state $\mathsf{2DH}_\mathsf{S}$, and receiving a message in the $\bot$ state leads to the state $\mathsf{2DH}_\mathsf{R}$. In this stage, the sending party Alice possesses Bob's public-key pk_R (and her own private state st_S and key sk_S). On the other hand, Bob only knows his private state st_B and key sk_B and obtains Alice's public key as part of her first message.

$\mathsf{2DH}_\mathsf{S}$ When a party is in the state $\mathsf{2DH}_\mathsf{S}$, this means that a $\mathsf{2DH}$ key is established to communicate with the receiving party. All ciphertexts so far have no forward security, as no ephemeral secrets of the receiving party are known to the sender.

$\mathsf{2DH}_\mathsf{R}$ A receiver in the state $\bot$ gets to the state $\mathsf{2DH}_\mathsf{R}$, which means that all incoming ciphertexts are encrypted without ephemeral secrets of the receiver and thus are not forward secure.

$\mathsf{4DH}$ When a sender in the state $\mathsf{2DH}_\mathsf{S}$ receives a message from the receiver, or the receiver in the $\mathsf{2DH}_\mathsf{R}$ state sends a message back to the sender, the ephemeral secrets of both parties can be used to encrypt messages. Therefore, they transition to the $\mathsf{4DH}$ state, meaning all exchanged ciphertexts in this state use ephemeral secrets of both parties and are, therefore, forward secure.

6.1.1 Intuitive Explanation for Establishment of a Channel

On a high level, a new channel is established in the following way: When Alice wishes to initiate a communication with Bob for the first time, then the local state of Alice with respect to Bob is $\perp$. This means that no ephemeral secret key has been established between both parties. Therefore, Alice samples an ephemeral key pair (x, X) and computes a $\mathsf{2DH}$ key $\mathsf{k}_{\mathsf{2DH}}$ using its own secret key, the public key of Bob and the ephemeral secret key x. This $\mathsf{2DH}$ key is used to derive a keychain key $\mathsf{k}_{\mathsf{KC,Alice}}$ and an encryption key $\mathsf{k}_{\mathsf{Enc,Alice}}$. The encryption key is used to encrypt the messages, and the keychain key is used to derive the next key of the keychain.

If Bob receives a ciphertext from Alice and the local state of Bob with respect to Alice is $\perp$, then Bob assumes this is a new communication and uses its secret key together with the public key and the learned ephemeral key X of Alice to derive the same keychain and encryption keys $\mathsf{k}_{\mathsf{KC,Alice}}$ and $\mathsf{k}_{\mathsf{Enc,Alice}}$.

When Bob answers Alice, he samples a fresh ephemeral key pair (y, Y) and computes a $\mathsf{4DH}$ key $\mathsf{k}_{\mathsf{4DH}}$ using the $\mathsf{2DH}$ key $\mathsf{k}_{\mathsf{2DH}}$ and his ephemeral secret key y, the public key of Alice and the ephemeral public key X of Alice. This $\mathsf{4DH}$ key has as input both parties' static and ephemeral keys due to four parallel Diffie-Hellman key exchanges. Furthermore, Bob answers with its ephemeral public key Y. After Alice receives this answer from Bob, both can communicate using a $\mathsf{4DH}$ key since both ephemeral keys are known to both parties.

6.2 Formal Description

Below we give a formal description of the sending and the receiving algorithm in Figure 10 and Figure 11, respectively. Since sending and receiving depends on the state in which the party is, we split up the definition for the sending and receiving algorithms per the possible states.

6.2.1 Helping Methods

To simplify the exposition of the protocol, we define several methods that are used within the protocol.

$Z \leftarrow \mathsf{DH}(x, y)$: Intuitively, this interface abstracts the main functionality of the Diffie-Hellman key exchange protocols as it takes as input some private exponent x and some public value y; it outputs $Z := y^x$.

$(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{KDF}(k, \mathsf{nonce})$: The method $\mathsf{KDF}(k, \mathsf{nonce})$ is a key derivation function (KDF) takes as input some high-entropy key k and some (possibly empty) string nonce. The output is a pair of independent keys $(\mathsf{k}_0, \mathsf{k}_1)$. The general usage of the function is the expansion of key material, such that multiple uniform keys can be obtained from a single source. Our formalization is an application-specific one, meaning that the outputs of KDFs, in general, may differ. The nonce often serves as a domain separator to make sure that certain keys are only used with a specific scope of the protocol.

$(x, X) \leftarrow \mathsf{Gen}_{\mathsf{eKE}}(\lambda)$: The input of the ephemeral key generation algorithm is the security parameter λ, it samples a uniformly chosen value $x \leftarrow\!\!\$\ \mathbb{Z}_p$, sets $X := g^x$, and outputs the pair (x, X).

6.2.2 Formal Description of the Protocol

The formalization of the Ibex protocol is shown in Figure 10 for the sending algorithms, and Figure 11 describes the algorithms of the receiver. Within this formalization, all private keys sk are elements of $\mathbb{Z}_p$, and the corresponding public keys are group elements. The protocol consists of the following steps:

Initialization The initialization of the protocol is shown in Lines 00 and 01. The first line expresses the parsing of the state $\mathsf{st}[\mathsf{pk}_R]$, which encodes in which local state the communication between the sender and receiver is. Depending on the local state, the corresponding algorithm is returned.

First sending round $\mathsf{send}_\perp(\mathsf{st}_S, \mathsf{sk}_S, \mathsf{pk}_R)$: This method is used to initialize the interaction with the receiver for the first time and works as follows. The sender samples an ephemeral key (Line 02, c.f. Section 6.2.1) and computes Diffie-Hellman key shares using DH in Line 03. Subsequently, it derives the keychain key $\mathsf{k}_{\mathsf{KC,Alice}}$ and the encryption key $\mathsf{k}_{\mathsf{Enc,Alice}}$ with the help of the key derivation function KDF. The sender stores the local values in its state (Line 05); it outputs the updated state, the encryption key $\mathsf{k}_{\mathsf{Enc},S}$, and the ciphertext $X\|\mathsf{pk}_S$ that is sent to the receiver.

First receiving round $\mathsf{recv}_\perp(\mathsf{st}_R, \mathsf{sk}_R, c)$: Once a ciphertext from a new sender arrives, the receiver parses the first message c as $X||\mathsf{pk}_S$, consisting of the "public part" of the Diffie-Hellman tuple and the sender's public key (c.f. Figure 11, Line 03). The receiver finishes this part of the protocol analogously to the sender by first computing the joint key in Line 04 and by deriving the keychain key $\mathsf{k}_{\mathsf{KC,Alice}}$ and the encryption key $\mathsf{k}_{\mathsf{Enc,Alice}}$ with the help of the key derivation function KDF in Line 05. The receiver stores the local values in its state (Line 06) and returns the updated state, the encryption key, and $\perp$.

Second sending round $\mathsf{send}_{\mathsf{2DH_S}}(\mathsf{st}_S, \mathsf{sk}_S, \mathsf{pk}_R)$: If the sender wishes to send more messages before the receiver responded to the first message, the sender parses its state $\mathsf{st}_S[\mathsf{pk}_R]$ as $(\mathsf{2DH_S}, \mathsf{k}_{\mathsf{2DH}}, \mathsf{k}_{\mathsf{KC},S}, x)$ (Line 07), it computes the ephemeral public key as X (Line 08). It updates the keychain with the help of the KDF in Line 09. Finally, the sender updates its internal state (Line 10) and outputs this value together with the encryption key and $X\|\mathsf{pk}_S$ (Line 11).

Second receiver round $\mathsf{recv}_{\mathsf{2DH_R}}(\mathsf{st}_R, \mathsf{sk}_R, c)$: The receiver upon receiving the ciphertext c and parsing it into $X\|\mathsf{pk}_S$ (Line 08), parses its state as $(\mathsf{2DH_R}, \mathsf{k}_{\mathsf{2DH}}, \mathsf{k}_{\mathsf{KC},S}, X)$ (Line 09). Moreover, it updates its local keychain in Line (10), as well as its state (Line 11), and returns these values at the end of this round (Line 12).

Third sending round $\mathsf{send}_{\mathsf{2DH_R}}(\mathsf{st}_S, \mathsf{sk}_S, \mathsf{pk}_R)$: The third sending round leads to a $\mathsf{4DH}$ secure communication. This means a message encrypted with the key derived in this sending round achieves forward security. In the first step, the sender parses its local state (Line 12), computes a fresh ephemeral key share (y, Y) (Line 13), and stores all values in $\mathsf{k}_{\mathsf{4DH}}$ (Line 14). It uses this key to derive two keychains, one for the sender (Line 15) and one for the receiver (Line 16). The sender updates its local state (Line 17) and outputs the corresponding values at the end of this round (Line 18). Note that the sender in this round is the initial receiver of the first message.

Third receiving round $\mathsf{recv}_{\mathsf{2DH_S}}(\mathsf{st}_R, \mathsf{sk}_R, c)$**:** The third receiving round is the first receive that uses $\mathsf{4DH}$. The receiver parses both the ciphertext c as $Y\|\mathsf{pk}_S$ (Line 13) and its local state $(\mathsf{2DH_S}, \mathsf{k_{2DH}}, \mathsf{k}_{\mathsf{KC},S}, x)$ (Line 14). It computes the joint key $\mathsf{k_{4DH}}$ (Line 15) and derives the key stream for the sender (Line 16) and the receiver (Line 17) based on this key. Finally, the receiver updates its state (Line 18) and returns the corresponding values (Line 19). Note that the receiver in this round is the initial sender of the first message.

Fourth sending round $\mathsf{send}_{\mathsf{4DH}}(\mathsf{st}_S, \mathsf{sk}_S, \mathsf{pk}_R)$**:** The last round of the protocol updates the sender's key stream and consists of the following steps. First, the sender parses its local state (Line 19) and updates the key stream (Line 20) and its state (Line 21), which now contains an updated keychain key. The new keys are derived using the key-derivation function KDF with the stored keychain key as input. Finally, it returns the corresponding values (Line 22).

Fourth receiving round $\mathsf{recv}_{\mathsf{4DH}}(\mathsf{st}_R, \mathsf{sk}_R, c)$**:** The receiver updates its keystream analogously to the fourth sending round, by performing the following step. It parses c as pk_S (Line 20), its state $(4DH, \mathsf{k}_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{KC},R})$ (Line 21), and uses the key derivation function to update its keychain $\mathsf{k}'_{\mathsf{KC},S}$ (Line 22) as well as its own state (Line 23). Finally, it outputs the corresponding values (Line 24).

```
**Protocol** $\mathsf{send}_{\mathsf{KE}}(\mathsf{st}_S, \mathsf{sk}_S, \mathsf{pk}_R)$
00 $(\mathsf{state}, \perp, \ldots, \perp) \leftarrow \mathsf{st}_S[\mathsf{pk}_R]$
01 Return $\mathsf{send}_{\mathsf{state}}(\mathsf{st}_S, \mathsf{sk}_S, \mathsf{pk}_R)$

$\mathsf{send}_{\perp}(\mathsf{st}_S, \mathsf{sk}_S, \mathsf{pk}_R)$
02 $(x, X) \leftarrow \mathsf{Gen}_{\mathsf{eKE}}(\lambda)$
03 $\mathsf{k}_{\mathsf{2DH}} \leftarrow \mathsf{DH}(\mathsf{sk}_S, \mathsf{pk}_R) \| \mathsf{DH}(x, \mathsf{pk}_R)$
04 $\mathsf{k}_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{Enc},S} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathsf{2DH}}, \text{'ke-2dh-<ID-S>'})$
05 $\mathsf{st}_S[\mathsf{pk}_R] \leftarrow (\mathsf{2DH_S}, \mathsf{k}_{\mathsf{2DH}}, \mathsf{k}_{\mathsf{KC},S}, x)$
06 Return $(\mathsf{st}_S, \mathsf{k}_{\mathsf{Enc},S}, X \| \mathsf{pk}_S)$

$\mathsf{send}_{\mathsf{2DH_S}}(\mathsf{st}_S, \mathsf{sk}_S, \mathsf{pk}_R)$
07 $(\mathsf{2DH_S}, \mathsf{k}_{\mathsf{2DH}}, \mathsf{k}_{\mathsf{KC},S}, x) \leftarrow \mathsf{st}_S[\mathsf{pk}_R]$
08 $X := g^x$
09 $\mathsf{k}'_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{Enc},S} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathsf{KC},S})$
10 $\mathsf{st}_S[\mathsf{pk}_R] \leftarrow (\mathsf{2DH_S}, \mathsf{k}_{\mathsf{2DH}}, \mathsf{k}'_{\mathsf{KC},S}, x)$
11 Return $(\mathsf{st}_S, \mathsf{k}_{\mathsf{Enc},S}, X \| \mathsf{pk}_S)$

$\mathsf{send}_{\mathsf{2DH_R}}(\mathsf{st}_S, \mathsf{sk}_S, \mathsf{pk}_R)$
12 $(\mathsf{2DH_R}, \mathsf{k}_{\mathsf{2DH}}, \mathsf{k}_S, X) \leftarrow \mathsf{st}_S[\mathsf{pk}_R]$
13 $(y, Y) \leftarrow \mathsf{Gen}_{\mathsf{eKE}}(\lambda)$
14 $\mathsf{k}_{\mathsf{4DH}} \leftarrow \mathsf{k}_{\mathsf{2DH}} \| \mathsf{DH}(y, \mathsf{pk}_R) \| \mathsf{DH}(y, X)$
15 $\mathsf{k}_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{Enc},S} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathsf{4DH}}, \text{'ke-4dh-<ID-S>'})$
16 $\mathsf{k}_{\mathsf{KC},R}, \mathsf{k}_{\mathsf{Enc},R} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathsf{4DH}}, \text{'ke-4dh-<ID-R>'})$
17 $\mathsf{st}_S[\mathsf{pk}_R] \leftarrow (\mathsf{4DH}, \mathsf{k}_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{KC},R})$
18 Return $(\mathsf{st}_S, \mathsf{k}_{\mathsf{Enc},S}, Y \| \mathsf{pk}_S)$

$\mathsf{send}_{\mathsf{4DH}}(\mathsf{st}_S, \mathsf{sk}_S, \mathsf{pk}_R)$
19 $(\mathsf{4DH}, \mathsf{k}_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{KC},R}) \leftarrow \mathsf{st}_S[\mathsf{pk}_R]$
20 $\mathsf{k}'_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{Enc},S} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathsf{KC},S})$
21 $\mathsf{st}_S[\mathsf{pk}_R] \leftarrow (\mathsf{4DH}, \mathsf{k}'_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{KC},R})$
22 Return $(\mathsf{st}_S, \mathsf{k}_{\mathsf{Enc},S}, \mathsf{pk}_S)$
```

Figure 10: Formalization of send in 4DH.

```
**Protocol** $\mathsf{recv}_{\mathsf{KE}}(\mathsf{st}_R, \mathsf{sk}_R, c)$
00 $\ldots \| \mathsf{pk}_S \leftarrow c$
01 $(\mathsf{state}, \perp, \ldots, \perp) \leftarrow \mathsf{st}_R[\mathsf{pk}_S]$
02 Return $\mathsf{recv}_{\mathsf{state}}(\mathsf{st}_R, \mathsf{sk}_R, c)$

$\mathsf{recv}_{\perp}(\mathsf{st}_R, \mathsf{sk}_R, c)$
03 $X \| \mathsf{pk}_S \leftarrow c$
04 $\mathsf{k}_{\mathsf{2DH}} \leftarrow \mathsf{DH}(\mathsf{sk}_R, \mathsf{pk}_S) \| \mathsf{DH}(\mathsf{sk}_R, X)$
05 $\mathsf{k}_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{Enc},S} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathsf{2DH}}, \text{'ke-2dh-<ID-S>'})$
06 $\mathsf{st}_R[\mathsf{pk}_S] \leftarrow (\mathsf{2DH_R}, \mathsf{k}_{\mathsf{2DH}}, \mathsf{k}_{\mathsf{KC},S}, X)$
07 Return $(\mathsf{st}_R, \mathsf{k}_{\mathsf{Enc},S}, \perp)$

$\mathsf{recv}_{\mathsf{2DH_R}}(\mathsf{st}_R, \mathsf{sk}_R, c)$
08 $X \| \mathsf{pk}_S \leftarrow c$
09 $(\mathsf{2DH_R}, \mathsf{k}_{\mathsf{2DH}}, \mathsf{k}_{\mathsf{KC},S}, X) \leftarrow \mathsf{st}_R[\mathsf{pk}_S]$
10 $\mathsf{k}'_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{Enc},S} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathsf{KC},S})$
11 $\mathsf{st}_R[\mathsf{pk}_S] \leftarrow (\mathsf{2DH_S}, \mathsf{k}_{\mathsf{2DH}}, \mathsf{k}'_{\mathsf{KC},S}, X)$
12 Return $(\mathsf{st}_R, \mathsf{k}_{\mathsf{Enc},S}, \perp)$

$\mathsf{recv}_{\mathsf{2DH_S}}(\mathsf{st}_R, \mathsf{sk}_R, c)$
13 $Y \| \mathsf{pk}_S \leftarrow c$
14 $(\mathsf{2DH_S}, \mathsf{k}_{\mathsf{2DH}}, \mathsf{k}_{\mathsf{KC},S}, x) \leftarrow \mathsf{st}_R[\mathsf{pk}_S]$
15 $\mathsf{k}_{\mathsf{4DH}} \leftarrow \mathsf{k}_{\mathsf{2DH}} \| \mathsf{DH}(\mathsf{sk}_R, Y) \| \mathsf{DH}(x, Y)$
16 $\mathsf{k}_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{Enc},S} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathsf{4DH}}, \text{'ke-4dh-<ID-S>'})$
17 $\mathsf{k}_{\mathsf{KC},R}, \mathsf{k}_{\mathsf{Enc},R} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathsf{4DH}}, \text{'ke-4dh-<ID-R>'})$
18 $\mathsf{st}_R[\mathsf{pk}_S] \leftarrow (\mathsf{4DH}, \mathsf{k}_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{KC},R})$
19 Return $(\mathsf{st}_R, \mathsf{k}_{\mathsf{Enc},S}, \perp)$

$\mathsf{recv}_{\mathsf{4DH}}(\mathsf{st}_R, \mathsf{sk}_R, c)$
20 $\mathsf{pk}_S \leftarrow c$
21 $(\mathsf{4DH}, \mathsf{k}_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{KC},R}) \leftarrow \mathsf{st}_R[\mathsf{pk}_S]$
22 $\mathsf{k}'_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{Enc},S} \leftarrow \mathsf{KDF}(\mathsf{k}_{\mathsf{KC},S})$
23 $\mathsf{st}_R[\mathsf{pk}_S] \leftarrow (\mathsf{4DH}, \mathsf{k}'_{\mathsf{KC},S}, \mathsf{k}_{\mathsf{KC},R})$
24 Return $(\mathsf{st}_R, \mathsf{k}_{\mathsf{Enc},S}, \perp)$
```

Figure 11: Formalization of recv in 4DH.

7 Security Proof

In this section, we formally show the security of Ibex. This means we claim the security of Ibex based on several assumptions in Theorem 2 and prove this theorem formally. The security of Ibex is based on the (established) assumptions that the Gap Diffie-Hellman assumption holds and the Key Derivation Function (KDF) is modeled as a random oracle. We discuss the meaningfulness of these assumptions in Section 7.1.

Theorem 2. *The Ibex protocol is a secure continuous key exchange protocol under the GDH assumption, assuming all KDFs are modeled as random oracles. That means if there exists an efficient adversary who can win the continuous key exchange experiment for the Ibex protocol with a non-negligible advantage* $\mathsf{Adv}_{\mathsf{Ibex}}(\lambda)$*, then there exists an efficient adversary who can break the* GDH *assumption with non-negligible probability* $\mathsf{Adv}_{\mathsf{GDH}}(\lambda)$*, where*

$$\mathsf{Adv}_{\mathsf{Ibex}}(\lambda) \leq \mathsf{n}_{chall} \cdot \mathsf{n}_{parties}^2 \cdot (\mathsf{Adv}_{\mathsf{GDH}}(\lambda) + \frac{\mathsf{n}_{ratchets}}{2^{\ell_{\mathsf{KDF}}}}).$$

The probabilities are taken over the random choices of all randomized algorithms, n_{chall} *denotes the number of challenges queried in the key exchange experiment,* $\mathsf{n}_{parties}$ *the number of spawned parties,* $\mathsf{n}_{ratchets}$ *the maximal number of key derivations, and* ℓ_{KDF} *the output size of the KDF random oracle.*

Before formally proving Theorem 2, we outline our proof strategy. We aim to show that the advantage of any efficient adversary winning the continuous key exchange experiment as in Figure 7 is a negligible function in the security parameter. One challenge in proving this theorem is that the adversary might request many challenges, i.e., the attacker might try to extract some information about the underlying keys by adaptively querying the corresponding challenge oracle. To handle this case, we will use a standard hybrid argument over the challenges queried by the adversary. A hybrid argument is a proof technique that allows reducing the case of *many* challenges to the case of a *single* challenge. In other words, seeing many challenges does not give the adversary a significant advantage. To apply this technique, for each fixed intermediate hybrid, we provide a series of game hops that modifies the experiment $\mathsf{Exp}^b_{\mathsf{KE},\mathcal{A}}(\lambda)$ towards a security experiment that allows breaking the GDH assumption whenever an efficient adversary wins $\mathsf{Exp}^b_{\mathsf{KE},\mathcal{A}}(\lambda)$. The initial game G_0 is a hybrid H_i. In the game G_1, we embed the challenge values of the GDH experiment into both the keys of the Sender and the keys of the Receiver. To simulate the hybrid H_i correctly, we program the KDF random oracle such that the reduction can answer all oracles queries of the adversary. Calculating the success probability of this game, we can argue that no efficient adversary against the security of Ibex can exist in the random oracle model if the GDH assumption holds.

Proof of Theorem 2. In this proof, we formally show that the advantage Adv_0 of any efficient adversary winning the continuous key exchange experiment as in Figure 7 instantiated with the Ibex protocol as depicted in Figure 10 and Figure 11 is a negligible function in the security parameter. To do so, we assume by contradiction that there exists an efficient adversary $\mathcal{A}$ that wins $\mathsf{Exp}^b_{\mathsf{KE},\mathcal{A}}(\lambda)$ with a non-negligible advantage. Using this adversary $\mathcal{A}$, we build a reduction from the experiment $\mathsf{Exp}^{b}_{\mathsf{KE},\mathcal{A}}(\lambda)$ to the experiment $\mathsf{GDH}_{\mathcal{A},\mathsf{GGen}}(\lambda)$. As we assume GDH to be hard, no such efficient adversary can exist, and thus the advantage Adv_0 is a negligible function in the security parameter λ.

To win the experiment $\mathsf{Exp}^{b}_{\mathsf{KE},\mathcal{A}}(\lambda)$, an adversary must correctly guess the used bit b, which is used in the challenge oracle. Here, each challenge key k_b output by the challenge oracle is either the currently used key in a session between two users i and j or a randomly chosen key. While running the experiment $\mathsf{Exp}^{b}_{\mathsf{KE},\mathcal{A}}(\lambda)$, we assume the adversary $\mathcal{A}$ queries exactly $\mathsf{n}_{\text{chall}}$ many challenge-keys k_b, where $\mathsf{n}_{\text{chall}} \in \mathsf{poly}(\lambda)$. Therefore, the adversary $\mathcal{A}$ wins the game $\mathsf{Exp}^{b}_{\mathsf{KE},\mathcal{A}}(\lambda)$ with a better probability than purely guessing when trying to determine the bit used in the set $\{k_b^1, \dots, k_b^{\mathsf{n}_{\text{chall}}}\}$ that equals either $\{k_0^1, \dots, k_0^{\mathsf{n}_{\text{chall}}}\}$ or $\{k_1^1, \dots, k_1^{\mathsf{n}_{\text{chall}}}\}$.

For these sets, which we refer to as the extreme hybrids, we build intermediate hybrids $H_\iota := \{k_1^1, \dots, k_0^\iota, k_0^{\iota+1}, \dots, k_0^{\mathsf{n}_{\text{chall}}}\}$, such that the extreme hybrid H_0 is the output of the challenge oracle in the game $\mathsf{Exp}^{0}_{\mathsf{KE},\mathcal{A}}(\lambda)$ and the hybrid $H_{\mathsf{n}_{\text{chall}}}$ is the output of the challenge oracle in the game $\mathsf{Exp}^{1}_{\mathsf{KE},\mathcal{A}}(\lambda)$.

By our hypothesis, the adversary $\mathcal{A}$ can distinguish the extreme hybrids with non-negligible advantage. Since the total number of intermediate hybrids is polynomial in λ, a non-negligible gap between the extreme hybrids translates into a non-negligible gap between a pair of neighboring intermediate hybrids. Therefore, our reduction can efficiently guess the position $\iota \in 1 \dots \mathsf{n}_{\text{chall}}$ of this gap with probability $\frac{1}{\mathsf{n}_{\text{chall}}}$ and we now build a series of game hops, where we consider one hybrid step H_ι at a time. Based on this, we now start a series of game hops with a negligible transition between neighboring games. Following this approach, we start with an initial game G_0 that equals the original game $\mathsf{Exp}^{0}_{\mathsf{KE},\mathcal{A}}(\lambda)$ and end with the original game $\mathsf{Exp}^{1}_{\mathsf{KE},\mathcal{A}}(\lambda)$. Doing so removes complexity from the proof and therefore makes the proof better understandable. Simultaneously, if we prove that the advantage of any adversary in winning G_0 is negligible, the same holds for the experiment $\mathsf{Exp}^{b}_{\mathsf{KE},\mathcal{A}}(\lambda)$. This follows from the standard hybrid argument as we model the hybrid H_ι in G_0. Therefore, a negligible advantage in winning an intermediate hybrid H_ι implies a negligible difference between the extreme hybrids H_0 and $H_{\mathsf{n}_{\text{chall}}}$.

<u>The Game G_0.</u> The initial game G_0 equals the intermediate hybrid H_ι. This means, the challenge oracle $\mathsf{Challenge}$ returns k_0 the first ι times and k_1 the last $\mathsf{n}_{\text{chall}} - \iota$ times. In a later game hop, we have to embed the challenge values of GDH into the keys of the members of session ι. Therefore, the hybrid step guesses the members in advance. This induces a loss of $\frac{1}{\mathsf{n}^2_{\text{parties}}}$ to the remaining advantage of an adversary against Ibex. A formal description of G_0 is given in Figure 12, and we mark all modifications in gray.

<u>The Game G_1.</u> The first game, G_1, embeds the challenges of the GDH experiment in the public keys of the sender and the receiver of the gap session. This gap session takes place between the parties i and j since the game hops happen in an intermediate hybrid that has a fixed gap session. For this session, we assume, without loss of generality, i initializes the communication. Furthermore, we denote (X, x) as the ephemeral key of i and (Y, y) as the ephemeral key of j for this gap session. The goal of G_1 is to embed the challenge values $[a], [b]$ of the GDH experiment into both the key-seed $\mathsf{k}_{\mathsf{4DH}}$ and the key-seed $\mathsf{k}_{\mathsf{2DH}}$ of this session which have the

```
Game $\mathsf{G}_0(\boxed{i, j}, \lambda)$
00 $\boxed{\mathsf{ctr}_{\text{chall}} \leftarrow 0}$
01 $n \leftarrow 0$
02 $b' \leftarrow \mathcal{A}(1^\lambda)$
03 Return $b'$

Oracle $\mathsf{Gen}(\lambda)$
04 $n \leftarrow n + 1$
05 $(\mathsf{sk}_n, \mathsf{pk}_n) \leftarrow \mathsf{Gen}_{\mathsf{KE}}(1^\lambda)$
06 Return $\mathsf{pk}_n$

Oracle $\mathsf{Send}(i, j)$
07 $(\mathsf{st}_i, \mathsf{k}, c) \leftarrow \mathsf{send}_{\mathsf{KE}}(\mathsf{st}_i, \mathsf{sk}_i, \mathsf{pk}_j)$
08 Return $(\mathsf{k}, c)$

Oracle $\mathsf{Challenge}(i, j)$
09 $\mathsf{ctr}_{\text{chall}} += 1$
10 $(\mathsf{st}_i, \mathsf{k}_0, c) \leftarrow \mathsf{send}_{\mathsf{KE}}(\mathsf{st}_i, \mathsf{sk}_i, \mathsf{pk}_j)$
11 $\mathsf{k}_1 \leftarrow \{0,1\}^\lambda$
12 $\boxed{\textbf{if } \mathsf{ctr}_{\text{chall}} < \iota : \text{ Return } (\mathsf{k}_0, c)}$
13 $\boxed{\textbf{if } \mathsf{ctr}_{\text{chall}} > \iota : \text{ Return } (\mathsf{k}_1, c)}$
14 Return $(\mathsf{k}_b, c)$

Oracle $\mathsf{Recv}(i, c)$
15 $(\mathsf{st}_i, \mathsf{k}, \mathsf{pk}) \leftarrow \mathsf{recv}_{\mathsf{KE}}(\mathsf{st}_i, \mathsf{sk}_i, c)$
16 If in-sync: Return $\mathsf{pk}$
17 Return $(\mathsf{k}, \mathsf{pk})$

Oracle $\mathsf{Corrupt}(i)$
18 Return $(\mathsf{sk}_i, \mathsf{st}_i)$
```

Figure 12: The initial game G_0.

following form.

$$\mathsf{k}_{\mathsf{2DH}} \leftarrow \mathsf{DH}(\mathsf{sk}_i, \mathsf{pk}_j) \| \mathsf{DH}(x, \mathsf{pk}_j)$$
$$\mathsf{k}_{\mathsf{4DH}} \leftarrow \mathsf{k}_{\mathsf{2DH}} \| \mathsf{DH}(\mathsf{sk}_i, Y) \| \mathsf{DH}(x, Y)$$

To embed the challenge values in the keys of the two parties i and j, the reduction must implement a case distinction that considers all options in which the two parties can be corrupted before and/or after computing the corresponding keys of the two parties. We model this case distinction using the two binary variables Corr_i and Corr_j that indicate whether party i or party j are corrupted during the execution of the experiment. This leads to four possible corruption states:

$\mathsf{Corr}_i \wedge \mathsf{Corr}_j$: In this case, the adversary eventually corrupts both parties i and j. Therefore, the reduction embeds the GDH challenge values into the ephemeral public keys X and Y.

$\mathsf{Corr}_i \wedge \neg\mathsf{Corr}_j$: In this case, the adversary eventually corrupts party i, but not party j. Thus, the reduction embeds the GDH challenge values into the ephemeral key X of party i and the static key pk_j of party j.

$\neg\mathsf{Corr}_i \wedge \mathsf{Corr}_j$: In this case, the adversary eventually corrupts party j, but not party i. Thus, the reduction embeds the GDH challenge values into the ephemeral key Y of party j and the static key pk_i of party i.

$\neg\mathsf{Corr}_i \wedge \neg\mathsf{Corr}_j$: In this case, the adversary does corrupt neither of both parties i and j. Therefore, the reduction embeds the GDH challenge values into the static public keys pk_i and pk_j.

Note, that these four different possibilities fully exhaust all possibilities. Since the reduction has a solution for all four cases, we can now assume, without loss of generality, that the reduction is in the right corruption state. For better readability, we formalize all four cases in G_1 unified. We fix whether a party is corrupted in the game execution at the beginning of G_1 via the variables Corr_i and Corr_j. If one of the two parties is corrupted during the execution of G_1, then the GDH challenge is embedded in the ephemeral key of this party. Else, it is inserted into the static key. Thus, we update the ephemeral and the static key-generation algorithm accordingly in Figure 13. These updated key-generation algorithms allow the reduction to answer to the oracle Corrupt consistently but raise a problem in computing the key-seed $\mathsf{k}_{\mathsf{4DH}}$ since, in each case, the reduction cannot compute at least one Diffie-Hellman tuple that corresponds to $\mathsf{DH}(a, B) = g^{ab}$. We address this issue by programming the key derivation function modeled as a random oracle.

<u>G_1 : Programming the KDF RO.</u> When the reduction has to compute $\mathsf{KDF}(\mathsf{k}, \mathsf{nonce})$ for key-seed k that should contain the Diffie-Hellman tuple $\mathsf{DH}(a, B)$, it runs the KDF on all computable group elements, as well as with the two group elements $[a]$ and $[b]$. When the reduction calls the KDF on these group elements and a nonce, the KDF returns two keys sampled uniformly at random and stores the query. On each query of $\mathcal{A}$, the KDF checks, if $\mathcal{A}$ queries KDF on at least one input $[ab]$ using the decisional Diffie-Hellman oracle $\mathcal{O}_{\mathsf{DDH}}$ of the underlying GDH game and aborts, if $\mathcal{A}$ does so. This abortion corresponds to the reduction winning the GDH experiment as it learns $[ab]$. Furthermore, as the game aborts, the programmed KDF answers consistently for each input if $\mathcal{A}$ asks for the programmed random value. A formal definition of G_1 is given in Figure 13.

```
**Game** $\mathsf{G}_1^{\mathcal{O}_{\mathsf{DDH}}}(\mathbb{G}, [1], [a], [b], i, \mathsf{Corr}_i, j, \mathsf{Corr}_j, \lambda)$
$\mathsf{ctr}_{\mathrm{chall}} \leftarrow 0$
$n \leftarrow 0$
$b' \leftarrow \mathcal{A}(1^\lambda)$
Return $b'$

**Oracle** $\mathsf{Gen}(\lambda)$
$n \leftarrow n + 1$
$(\mathsf{sk}_n, \mathsf{pk}_n) \leftarrow \mathsf{Gen}_{\mathsf{KE}}(1^\lambda)$
**if** $n = i : \wedge \neg \mathsf{Corr}_i :\ \mathsf{pk}_n = [a]$
**if** $n = j : \wedge \neg \mathsf{Corr}_i :\ \mathsf{pk}_n = [b]$
Return $\mathsf{pk}_n$

$\mathsf{Gen}_{\mathsf{eKE}}(\lambda, S, R)$
$(\mathsf{sk}_e, \mathsf{pk}_e) \leftarrow \mathsf{Gen}_{\mathsf{eKE}}(1^\lambda)$
**if** $S = i \wedge R = j$ :
    **if** $\mathsf{Corr}_i :\ \mathsf{pk}_e = [a], \mathsf{sk}_e = \perp$
    **if** $\mathsf{Corr}_j :\ \mathsf{pk}_e = [b], \mathsf{sk}_e = \perp$
Return $(\mathsf{sk}_e, \mathsf{pk}_e)$

**Oracle** $\mathsf{Send}(i, j)$
$(\mathsf{st}_i, \mathsf{k}, c) \leftarrow \mathsf{send}_{\mathsf{KE}}(\mathsf{st}_i, \mathsf{sk}_i, \mathsf{pk}_j)$
Return $(\mathsf{k}, c)$

**Oracle** $\mathsf{Challenge}(i, j)$
$\mathsf{ctr}_{\mathrm{chall}} += 1$
$(\mathsf{st}_i, \mathsf{k}_0, c) \leftarrow \mathsf{send}_{\mathsf{KE}}(\mathsf{st}_i, \mathsf{sk}_i, \mathsf{pk}_j)$
$\mathsf{k}_1 \leftarrow \{0,1\}^\lambda$
**if** $\mathsf{ctr}_{\mathrm{chall}} < \iota$ :  Return $(\mathsf{k}_0, c)$
**if** $\mathsf{ctr}_{\mathrm{chall}} > \iota$ :  Return $(\mathsf{k}_1, c)$
Return $(\mathsf{k}_b, c)$

**Oracle** $\mathsf{Recv}(i, c)$
$(\mathsf{st}_i, \mathsf{k}, \mathsf{pk}) \leftarrow \mathsf{recv}_{\mathsf{KE}}(\mathsf{st}_i, \mathsf{sk}_i, c)$
If *in-sync*: Return $\mathsf{pk}$
Return $(\mathsf{k}, \mathsf{pk})$

**Oracle** $\mathsf{Corrupt}(i)$
Return $(\mathsf{sk}_i, \mathsf{st}_i)$

$\mathsf{KDF}(g_1, g_2, g_3, g_4, \mathsf{nonce})$
**for** $x \in 1, \ldots, 4$ **do**
    **if** $g_x = \mathcal{O}_{\mathsf{DDH}}([a], [b])$ **abort**
Return $\mathsf{KDF}(g_1, g_2, g_3, g_4, \mathsf{nonce})$
```

Figure 13: The first game G_1.

Game G_1 simulates G_0 perfectly if it does not abort; this means all changes made in G_1 compared to G_0 cannot be detected by any efficient adversary if the game does not abort in line 27. This is the case, as the GDH challenge values are equally distributed like the $\mathsf{Gen}_{\mathsf{KE}}$ and $\mathsf{Gen}_{\mathsf{eKE}}$ values. If the game aborts in line 27, the reduction wins the GDH experiment, as we show in the following claim. Therefore,

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{G}_0}(\lambda) \leq \mathsf{Adv}^{\mathcal{A}}_{\mathsf{G}_1}(\lambda) + \Pr[\mathsf{break}_{\mathsf{GDH}}].$$

Claim 1. *Let* $\mathsf{break}_{\mathsf{GDH}}$ *be the event where game* G_1 *aborts in line 27. If* $\mathsf{break}_{\mathsf{GDH}}$ *happens, the reduction successfully finds a solution for the* GDH *experiment.*

Proof. As the KDF is modeled as a random oracle, the reduction learns all inputs to KDF. The KDF aborts in Line 27, if $g_x = g^{ab}$. Therefore, the input g_x to the KDF is a valid solution for the GDH challenge-instance. □

Game G_1 aborts if the adversary $\mathcal{A}$ finds a solution for the GDH experiment. Thus, the adversary $\mathcal{A}$ cannot win G_1 if it submits a solution for the GDH experiment to the KDF oracle. As the KDF is modeled as a random oracle, the challenge keys k_0 and k_1 are only distinguishable if the adversary $\mathcal{A}$ queries the KDF random oracle with a previous ratchet-key. This means the adversary $\mathcal{A}$ must guess such a previous ratchet-key to obtain an advantage in the game G_1.

Claim 2. *The advantage of* $\mathcal{A}$ *in winning game* G_1 *is bounded by the output size* ℓ_{KDF} *of the* KDF *and the number* $\mathsf{n}_{ratchets}$ *of messages the party* j *has received from party* i. *More formally,*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{G}_1}(\lambda) \leq \frac{\mathsf{n}_{ratchets}}{2^{\ell_{\mathsf{KDF}}}}.$$

Proof. Since the KDF is modeled as a random oracle, the only way to distinguish a random string $\mathsf{k}_1 \leftarrow \{0,1\}^\lambda$ from the output of the KDF on input x is by querying the KDF on x. If we assume the challenge key is the $\mathsf{n}_{\text{ratchets}}$-th derivation from the key-seed, the probability of guessing a random value x that is part of the challenge-key-ratchet is bounded by $\frac{\mathsf{n}_{\text{ratchets}}}{2^{\ell_{\mathsf{KDF}}}}$. □

Computing the Success Advantage. Using the standard hybrid argument, we have a loss of the factor $\mathsf{n}_{\text{chall}}$ due to the guessing of the gap session index. Furthermore, the reduction wins the GDH experiment if the guess of i and j is correct, which introduces a loss of the factor $\mathsf{n}^2_{\text{parties}}$. This leads to the following advantage term:

$$\begin{aligned}\mathsf{Adv}_{\text{Ibex},\mathcal{A}}(\lambda) &\leq \mathsf{n}_{\text{chall}} \cdot \mathsf{n}^2_{\text{parties}} \cdot \mathsf{Adv}_{\mathsf{G}_0,\mathcal{A}}(\lambda) \\ &= \mathsf{n}_{\text{chall}} \cdot \mathsf{n}^2_{\text{parties}} \cdot \mathsf{Adv}_{\mathsf{G}_1,\mathcal{A}}(\lambda) \\ &\leq \mathsf{n}_{\text{chall}} \cdot \mathsf{n}^2_{\text{parties}} \cdot \left(\frac{\mathsf{n}_{\text{ratchets}}}{2^{\ell_{\mathsf{KDF}}}} + \Pr[\mathsf{break}_{\mathsf{GDH}}]\right).\end{aligned}$$

Due to the GDH assumption, the advantage of any adversary in breaking the Ibex protocol $\mathsf{Adv}_{\text{Ibex},\mathcal{A}}(\lambda)$ is polynomially bounded by a negligible function. Therefore, no efficient adversary against Ibex can exist. □

7.1 Discussion of Assumptions

The core assumption made in our proof is that the Gap-Diffie-Hellman (GDH) problem is hard for the used (elliptic curve) group in the random oracle model (ROM). Analyses of similar protocols either follow the same approach (e.g., in [12, 13, 14, 11]). Alternatively (e.g., in [15, 16, 17]), it is assumed that the composition of Diffie-Hellman key exchange in the (elliptic curve) group and subsequent key derivation from the resulting key (via a key derivation function, hash function, or pseudo-random function) fulfills the PRF-ODH[3] assumption [18]. For details about many variants of the PRF-ODF assumption and an extensive discussion about its relation to the above-mentioned alternative via GDH and ROM, we refer the interested reader to the work by Brendel et al. [18].

While the ROM is established in analyses of real-world protocols, it is important to mention that it is an idealization that cannot be instantiated in the real world. Thus, although using the ROM simplifies protocol analyses and tames their complexity, in theory, cryptographers aim to avoid relying on the ROM whenever analyses without it lead to meaningful results. Nevertheless, in our concrete example, the only known instantiation of the PRF-ODH assumption is based on the GDH assumption via the ROM. Thus, we consider avoiding the ROM by only hiding it using the PRF-ODH assumption of little value.

We mainly chose to rely on GDH and ROM directly because the Diffie-Hellman key exchanges embedded in the 2DH and 4DH protocols do not align with the PRF-ODH assumption. Therefore, for our analysis, we would have needed to introduce new, untested variants of the PRF-ODH assumption, or we would have needed to consider variants of 2DH and 4DH that do not align with what is implemented and used in practice.

As mentioned, the ROM is established in the cryptographic literature because it supports clarity of security proofs. Taking all above aspects together, we believe our approach is the most suitable for our analysis. Nevertheless, we believe a similar result can be obtained by following an alternative path.

8 Practical Instantiation

In Theorem 2, we assume the used key-derivation function of Ibex to be instantiated by a random oracle. Yet, in the real protocol, Ibex uses BLAKE2b once in a keyed version and once in an unkeyed version. Therefore, in this section, we show that Theorem 2 carries over to the practical Ibex construction by showing that the keyed BLAKE2b is a pseudorandom function and the unkeyed BLAKE2b achieves indifferentiability.

Unkeyed BLAKE2b. Since the unkeyed BLAKE2b has no hidden information, it can not be shown to be a PRF. Therefore, we show the indifferentiability of the unkeyed BLAKE2b. The notion of indifferentiability was introduced by Maurer et al. in TCC 2004 [7] and adopted by Coron et al. in Crypto 2005 for the context of hash functions [6]. At a high-level, indifferentiability measures how close a hash function behaves as a random oracle. More formally:

[3]Short for Pseudo-Random Function Oracle-Diffie-Hellman.

Definition 10 ($\varepsilon-$indifferentiability of hash functions [8])**.** Let F be a hash function based on an ideal primitive f and R be a random oracle, and S be a simulator with access to R. We say that F^f is $\varepsilon-$indifferentiable from R if for any adversary $\mathcal{D}$, there exists a simulator $\mathcal{S}$ such that the advantage of the adversary is bound by

$$\mathsf{Adv}^{\mathsf{indiff}}_{F^f,S^R}(D) = |\Pr[\mathcal{D}^{F,f} = 1] - \Pr[\mathcal{D}^{R,S} = 1]| \leq \varepsilon.$$

The work of Luykx et al. [9] shows that the BLAKE2 compression function is indifferentiable to a random compression function.

Lemma 2.1 (Indifferentiability of BLAKE2 Compression Function). *The BLAKE2 compression function is indifferentiable from a random compression function up to about* $2^{\frac{n}{2}}$ *queries under the assumption that the underlying block cipher is randomly drawn [9].*

The indifferentiability of the BLAKE2 compression function justifies the usage of the RO instead of BLAKE2b in Theorem 2, since the following theorem holds.

Theorem 3. *Let C be a cryptosystem with oracle access to an ideal primitive F. Let F be an algorithm such that F based on f is indifferentiable from the random oracle R. Then the cryptosystem C is at least as secure in the f-model with algorithm F as in the R-model [6, 7].*

Keyed BLAKE2b. Since the unkeyed BLAKE2b is indifferentiable to a random oracle, it can be shown that BLAKE2b$(k, \cdot)$ for a random key k is a pseudorandom function [9].

Definition 11 (Pseudorandom Function)**.** Let $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$ be an efficient, length-preserving, keyed function. F is a *pseudorandom function (PRF)* if for all PPT distinguishers D, there exists a negligible function negl such that

$$\left|\Pr[D^{F(k,\cdot)}(1^\lambda) = 1] - \Pr[D^{f(\cdot)}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where the first probability is taken over the uniform choice of $k \in \{0,1\}^\lambda$ and the randomness of D, and the second probability is taken over the uniform choice of $f \in Func_n$ and the randomness of D.

Combination of unkeyed and keyed BLAKE2b. The Ibex protocol does not use a single execution of BLAKE2b as a KDF but a nested combination of both that is defined via

$$\mathsf{KDF}(g_1, g_2, g_3, g_4, \mathsf{nonce}) = \mathsf{BLAKE2b}(\mathsf{BLAKE2b}(g_1, g_2, g_3, g_4), \mathsf{nonce}).$$

The inner BLAKE2b execution hashes the four group elements to a 64-bit value which is used as a key for the keyed outer BLAKE2b instance. Since the unkeyed BLAKE2b achieves indifferentiability, this hashed key is indistinguishable from random as long as at least one of the four group elements is unknown to any distinguisher [9]. Therefore, by the PRF security of the keyed outer BLAKE2b, the nested construction is indistinguishable from a random function as long as at least one of the four group elements is unknown to any distinguisher. This justifies the usage of the random oracle instead of the nested BLAKE2b in Theorem 2.

9 Limitations of Our Analysis

While our security analysis encompasses various aspects, there are certain conditions under which the analysis may not hold unconditionally.

- Firstly, the Ibex protocol operates in a "double encryption" setup where the ciphertext is encrypted under the static key pair. This introduces a level of circularity not covered by our analysis. Proving circular security is a challenging task in general, and it was demonstrated by Boneh, Halevi, Hamburg, and Ostrovsky in 2008 that one-way security does not imply circular security [19].
- Secondly, our analysis assumes that the contacts' public keys have either been verified out-of-band (either by scanning the other parties' QR codes, manual comparison or by a Threema Work administrator), or the public key infrastructure (PKI) is honest. The case of unverified public keys and a compromised PKI has not been accounted for as it would introduce additional complexity.
- Third, Threema also supports a desktop application connecting to the smartphone version. The user reads a QR code to connect both applications and may secure the access to the desktop using a password. The security and privacy in this setting is also not part of this analysis.

10 Recommendations for Further Development

Post Quantum Security. Rapid advances in quantum computing pose a significant threat to the security of current cryptographic systems, including key exchange protocols. Quantum computers have the potential to efficiently solve certain mathematical problems that are currently intractable for classical computers. In particular, they can rapidly compute discrete logarithms, which is the basis of widely used cryptographic algorithms such as El-Gamal and Diffie-Hellman. As quantum computers become more powerful, traditional key exchange protocols based on Diffie-Hellman assumptions will be vulnerable to attacks. Given this impending challenge, it would be beneficial to invest in developing post-quantum secure key exchange protocols.

Group Key Management. The current strategy for group key exchange of Threema utilizes a one-to-one key exchange mechanism. Threema uses this approach to mitigate the leaking of a potential group's existence and structure. Yet, this approach falls short if an adversary can analyze the communication pattern of messages exchanged via the Threema server. Instead, we propose to use a different approach proposed in [21] that anonymizes the sender and the receiver of a message from the server's point of view. Further techniques to hide the existence of a group (and possibly its size) w.r.t. a malicious server might require the integration of techniques similar to oblivious RAM (ORAM) [22]. At a conceptual level, ORAM is a cryptographic mechanism that obfuscates read and write actions performed on a database maintained by an untrustworthy server. We propose investigating this avenue to augment users' privacy within a group context.

Downgrade Attacks. In situations where two users' protocol versions do not allow the Ibex protocol to be used, the current fallback option is to notify the user in the UI and revert to the old protocol, which does not achieve forward security. However, this approach is vulnerable to tampering attacks on the directory server. In detail, an adversary could modify the feature mask on the directory server to bypass Ibex and use the fallback protocol. Therefore, we propose to embed the protocol version in an associated data field to mitigate such attacks. By incorporating this enhancement, Ibex can ensure the integrity of communications during fallback scenarios and maintain secure and reliable communications.

Penetration Testing. It is important to note that our security analysis does not evaluate the specific implementation of the Ibex protocol in Threema. The analysis focuses on the protocol design and its theoretical/conceptual security properties. However, a comprehensive evaluation of the actual implementation, including vulnerability assessments and penetration testing, should be conducted by qualified penetration testers. Evaluating the implementation ensures that any potential implementation-specific flaws or vulnerabilities are identified and addressed, providing a more holistic assessment of the overall security of Threema's messaging platform.

11 Conclusion

This work represents a significant milestone as it provides the first formal security analysis of the Ibex protocol. In the past, theoretical security analyses have proven their effectiveness and value in the field of cryptography and information security. Theoretical security analyses are essential for evaluating the strength and reliability of cryptographic protocols and systems. By subjecting these protocols to rigorous examination, researchers can identify potential vulnerabilities, analyze their security properties, and assess their resilience against various types of attacks. Through these analyses, theoretical models can provide insights into the underlying security assumptions, threat models, and possible weaknesses in the design or implementation of protocols. Through a comprehensive examination, we have successfully evaluated the protocol's adherence to its promised security properties. The results of our analysis reveal that the Ibex protocol satisfactorily meets the desired security requirements. Notably, our investigation did not uncover any flaws or vulnerabilities within the protocol. This outcome underscores the sound cryptographic design of the Ibex protocol, as it has withstood a rigorous security analysis, and we were able to prove the security of Ibex formally. The absence of significant weaknesses further strengthens the confidence in the protocol's security guarantees and enhances its overall reliability.

References

[1] J. Katz and Y. Lindell, Introduction to Modern Cryptography, Chapman and Hall/CRC, 2014

[2] W. Diffie and M. Hellman, New directions in cryptography, IEEE Transactions on Information Theory, 22 (6), 1976, pp. 644-654.

[3] Barbara Guttman and E Roback, An Introduction to Computer Security: The NIST Handbook, 1995, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD

[4] Poettering, B., Rösler, P.: Combiners for AEAD. IACR Trans. Symm. Cryptol. 2020(1), 121–143 (2020). `https://doi.org/10.13154/tosc.v2020.i1.121-143`

[5] T. Okamoto, D. Pointcheval (2001). The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes. In: Kim, K. (eds) Public Key Cryptography. PKC 2001. Lecture Notes in Computer Science, vol 1992. Springer, Berlin, Heidelberg. `https://doi.org/10.1007/3-540-44586-2_8`

[6] J. S. Coron, Y. Dodis, C. Malinaud, and P. Puniya, Merkle-Damgard Revisited: How to Construct a Hash Function, Advances in Cryptology – CRYPTO'05, LNCS 3621, Springer-Verlag, pp. 430-448, 2005.

[7] U. Maurer, R. Renner, and C. Holenstein, Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology, TCC'04, , LNCS 2951, Springer-Verlag, pp. 21-39, 2004.

[8] D. Chang, M. Nandi, and M. Yung. Indifferentiability of the Hash Algorithm BLAKE, Cryptology ePrint Archive, Paper 2011/623.

[9] A. Luykx, B. Mennink, Bart, and S. Neves. (2016). Security Analysis of BLAKE2's Modes of Operation. IACR Transactions on Symmetric Cryptology. 158-176. `10.46586/tosc.v2016.i1.158-176`.

[10] M. Bellare and P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols, ACM CCS 1993. Full version available at http://cseweb.ucsd.edu/∼mihir/papers/ro.html.

[11] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. (2017). A Formal Security Analysis of the Signal Messaging Protocol. IEEE EuroS&P 2017

[12] M. Fischlin and F. Günther. (2014). Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. ACM CCS 2014.

[13] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru. How Secure and Quick is QUIC? Provable Security and Performance Analyses. IEEE Symposium on Security and Privacy, SP 2015.

[14] H. Krawczyk and H. Wee. (2016). The OPTLS Protocol and TLS 1.3. IEEE EuroS&P 2016.

[15] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. (2012). On the Security of TLS-DHE in the Standard Model. IACR CRYPTO 2012.

[16] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. (2015). A Cryptographic Analysis of the TLS 1.3 Handshake Protocol Candidates. ACM CCS 2015.

[17] B. Dowling, P. Rösler, and J. Schwenk. (2020). Flexible Authenticated and Confidential Channel Establishment (fACCE): Analyzing the Noise Protocol Framework. IACR PKC 2020.

[18] J. Brendel, M. Fischlin, F. Günther, and C. Janson. (2017). PRF-ODH: Relations, Instantiations, and Impossibility Results. IACR CRYPTO 2017.

[19] D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky (2008). Circular-Secure Encryption from Decision Diffie-Hellman. IACR CRYPTO 2008.

[20] K. Cohn-Gordon, C. Cremers, and L. Garratt. On Post-compromise Security. IEEE 29th Computer Security Foundations Symposium, CSF 2016.

[21] A. Bienstock, P. Rösler, and Y. Tang. ASMesh: Anonymous and Secure Messaging in Mesh Networks Using Stronger, Anonymous Double Ratchet. ACM CCS 2023.

[22] Stefanov, Emil and Dijk, Marten Van and Shi, Elaine and Chan, T.-H. Hubert and Fletcher, Christopher and Ren, Ling and Yu, Xiangyao and Devadas, Srinivas: Path ORAM: An Extremely Simple Oblivious RAM Protocol 2018, Association for Computing Machinery, J. ACM 2018.