Mind the Gap: Modular Machine-checked Proofs of One-Round Key Exchange Protocols

Gilles Barthe[1], Juan Manuel Crespo[1,2],
Yassine Lakhnech[3], and Benedikt Schmidt[1]

[1] IMDEA Software Institute, Spain
{gilles.barthe,benedikt.schmidt}@imdea.org
[2] FireEye Germany
juanmanuel.crespo@fireeye.com
[3] U. de Grenoble & VERIMAG, France
yassine.lakhnech@imag.fr

Abstract. Using EasyCrypt, we formalize a new modular security proof for one-round authenticated key exchange protocols in the random oracle model. Our proof improves earlier work by Kudla and Paterson (ASIACRYPT 2005) in three significant ways: we consider a stronger adversary model, we provide support tailored to protocols that utilize the Naxos trick, and we support proofs under the Computational DH assumption not relying on Gap oracles. Furthermore, our modular proof can be used to obtain concrete security proofs for protocols with or without adversarial key registration. We use this support to investigate, still using EasyCrypt, the connection between proofs without Gap assumptions and adversarial key registration. For the case of honestly generated keys, we obtain the first proofs of the Naxos and Nets protocols under the Computational DH assumption. For the case of adversarial key registration, we obtain machine-checked and modular variants of the well-known proofs for Naxos, Nets, and Naxos+.

Keywords: Provable Security; Security Protocols; EasyCrypt; Key Exchange; Interactive Theorem Proving

1 Introduction

Cryptographic protocols, like TLS, SSH, and VPNs, are one of the main building blocks of the Internet. At the heart of these protocols lies a key exchange protocol, which allows two parties to establish a shared session key used for building a secure channel. Traditionally, key exchange has often been realized using key transport protocols. Here, one participant generates the session key and uses public key encryption and signatures to transport it to the peer. Since this approach usually uses a longterm public key for encryption, it lacks resilience against leakage of the corresponding secret key, either through cryptanalysis or coercion. Concretely, if an adversary obtains the longterm secrets of a participant, he can obtain all his session keys. Resilience against such attacks is called

forward secrecy [24]. While long known in the cryptographic community, forward secrecy has recently come under public light following revelations about mass surveillance and implementation bugs such as Heartbleed. As a consequence, we expect that the ongoing shift from key transport protocols to key agreement protocols that achieve forward secrecy will accelerate; for instance, there is consensus to deprecate RSA key transport in TLS 1.3.

One solution to achieve forward secrecy is to use protocols that use an ephemeral Diffie-Hellman (DH) exchange. Since the ephemeral DH exchange uses fresh exponents for each session, protocols using them can provide forward secrecy. In order to provide authentication, most popular protocols such as TLS and SSH sign the exchanged DH messages. Theoretically, key agreement protocols based on signed DH are well understood and allow for relatively straightforward proofs of the classical security properties and forward secrecy [10,18]. In practice, their usage in real-world protocols poses additional problems and there is a large body of work on analyzing the security of the combined channel establishment protocol [27,32,13].

Nevertheless, the use of signatures has several disadvantages. First, standardization and implementation must include a signature scheme which might not be required otherwise. Second, the use of signatures might compromise deniability [30]. Third, signing and verification time might be a bottleneck. Furthermore, several realistic attacks are still possible for one-round versions of such protocols. For example, leakage of session randomness can lead to the compromise of future sessions in signed DH protocols [31, Section 1.6].

To address these deficiencies, implicitly authenticated key exchange (IAKE) protocols have been introduced in [40]. Such protocols enhance an ephemeral DH exchange with static DH keys that are only used in the key computation. Many protocols of this type have been proposed, such as HQMV [31], Naxos [36], and Nets [38], and they often surpass signature-based protocols in terms of performance and security. For example, the HMQV protocol, which is a hashed variant of the MQV [37] protocol, adds authentication to the ephemeral Diffie-Hellman protocol at a very low cost if Shamir's trick [25] is used for multi-exponentiation. Prominent instances of deployed systems based on such protocols include the EMV [17] chip based payment system, which uses a custom protocol and Blackberry phones, which use the elliptic curve version of MQV [37]. One of the main adversary models for IAKE protocols is the extended Canetti-Krawczyk (eCK) model [36], which provides very strong security guarantees such as (weak) perfect forward secrecy and session key secrecy even in the case where the session's randomness is leaked.

However, a number of concerns with the provable security of this class of protocols remain. First, only some of them achieve efficient designs and tight reductions under standard assumptions such as computational DH (CDH). Instead, known proofs of efficient protocols often use the Forking Lemma (and therefore give non-tight reductions), or strong assumptions such as Gap-CDH [43]. Second, and probably more importantly, the security definitions for key exchange protocols are an order of magnitude more complex than standard definitions for

most cryptographic primitives, such as IND-CCA. This results in long proofs that few people understand or check for flaws. Unsurprisingly, numerous attacks have been discovered on key exchange protocols [28,31,41,42], even on those claimed provably secure. This second problem is not exclusive to key exchange protocols. In fact, two approaches have been developed to tame the complexity of cryptographic proofs in the computational model.

The first approach is to develop generic results that can be applied to many concrete instances. While genericity does not eliminate the possibility of flaws, it allows to build a reduced corpus of results on which the security of protocols depends, and gives greater incentive to examine their proofs carefully. One popular class of generic results in cryptography are protocol transformations. If a protocol Π is secure with respect to an adversary model $\mathcal{M}$, then Π can be transformed into a (more complicated) protocol Π' that is secure with respect to a stronger adversary model $\mathcal{M}'$. For key exchange, this approach was pioneered by Bellare, Canetti, and Krawczyk [8] and other transformations have been proposed by Kudla and Paterson [33], Cremers and Feltz [23], and Boyd et al. [16]. However, existing transformations have several drawbacks, in particular: the transformation in [8] cannot be applied to many protocols of interest; the transformations in [23,16] assume that the initial protocol is already secure in the eCK model; and the transformation in [33] only supports proofs under Gap assumptions, predates the eCK model and is only applicable to weaker security models.

The second approach is to build machine-checked, independently verifiable proofs of security; this approach has been suggested notably by Halevi [26], and more recently by Hales[4] in the context of verifying the absence of trapdoors in NIST standards. Assuming that the verification tool is correct, one can gain trust in a formal proof simply by checking the definitions it uses and the theorem statement, since the tool ensures the correctness of the reasoning steps. There are two mature tools to perform machine-checked cryptographic proofs in the computational model: CryptoVerif [14] and EasyCrypt [6,5]. CryptoVerif is an automatic prover in the computational model and has been applied to cryptographic constructions such as the Full Domain Hash signature scheme, Kerberos, and the One-Encryption Key Exchange. EasyCrypt is a toolset for the construction and verification of game-based cryptographic proofs and has mostly been applied to cryptographic primitives, such as the Cramer-Shoup encryption scheme, and the OAEP padding scheme. So far neither of these tools have been used to obtain machine-checked proofs of modern key exchange protocols with respect to their intended security definitions.

Both approaches are complementary. Indeed, machine-checked proofs make *checking proofs* efficient, but they also significantly increase the cost of *building proofs*. As a consequence, generic results are ideal targets for machine-checked proofs, for two reasons. First, the cost of building proofs for generic results is justified by their multiple applications. Second, the level of abstraction required

[4] `https://jiggerwit.wordpress.com/2013/11/04/formalizing-nist-standards/`

to obtain generic proofs combined with the explicit tracking of assumptions in machine-checked proofs often provides new insights.

Contributions

We develop a new generic proof of security for key-exchange protocols, and instantiate it to obtain security proofs for known protocols with respect to different adversary models and hardness assumptions. In the cases of Naxos and Nets, we show that it is possible to obtain a CDH proof (without GAP) if static keys are honestly generated. We also formalize our generic proof and its instantiations using EasyCrypt. We elaborate on these points below.

Generic Proof for eCK security. We consider the class of one-round Diffie-Hellman protocols defined in the random oracle model where the session key is the output of a hash function. We reduce eCK-security of a key exchange protocol in this class to a condition on the key computation function and four simple games, in which the adversary can access at most one oracle. For protocols that employ the Naxos trick and use $h(x, a)$ as the exponent of the DH message, we provide an even simpler reduction with three games.

Concretely, we structure our generic proof in terms of protocol transformations and different versions of the security game. We are interested in eCK security. As proof tools, we also use three additional security games:

- eCK: Adversary must distinguish the session key of a fresh test session from random key.
- $\mathsf{eCK}^{\mathsf{nt}}$: Variant of eCK where adversary must provide the actor's static secret key as input to the ephemeral reveal oracle.
- CSK: Simplified game for protocols that do not use the Naxos trick where adversary must compute session key of test session (4 cases).
- $\mathsf{CSK}^{\mathsf{nt}}$: Simplified game for protocols that use the Naxos trick where adversary must compute session key of test session (3 cases).

We then define protocol transformations $\mathcal{T}^{\mathsf{nt}}$ (use Naxos trick) and $\mathcal{T}^{\mathsf{hsk}}$ (hash session key) and prove that the following implications hold for all protocols Π:

$$\begin{aligned}
\Pi \text{ is } \mathsf{eCK}^{\mathsf{nt}}\text{-secure} &\Longrightarrow \mathcal{T}^{\mathsf{nt}}(\Pi) \text{ is } \mathsf{eCK}\text{-secure} \\
\Pi \text{ is } \mathsf{CSK}^{\mathsf{nt}}\text{-secure} &\Longrightarrow \mathcal{T}^{\mathsf{hsk}}(\Pi) \text{ is } \mathsf{eCK}^{\mathsf{nt}}\text{-secure} \\
\Pi \text{ is } \mathsf{CSK}\text{-secure} &\Longrightarrow \mathcal{T}^{\mathsf{hsk}}(\Pi) \text{ is } \mathsf{eCK}\text{-secure}
\end{aligned}$$

As an example, consider the Naxos protocol which uses the Naxos trick and hashes its session key. We first define the "core" of Naxos and call it Naxos^{core}. Since it holds that $\mathsf{Naxos} = \mathcal{T}^{\mathsf{hsk}}(\mathcal{T}^{\mathsf{nt}}(\mathsf{Naxos}^{core}))$, it suffices to prove that Naxos^{core} is $\mathsf{CSK}^{\mathsf{nt}}$-secure to obtain that Naxos is eCK-secure. While the original eCK security definition consists of a game with seven oracles where the winning condition contains a complicated freshness condition, the $\mathsf{CSK}^{\mathsf{nt}}$ game has a very simple winning condition and only provides a decision oracle that allows the adversary to confirm session key guesses.

Protocol	Existing Proof	Our Proofs	EasyCrypt
Naxos [36]	eCK/Gap-CDH	$\mathsf{eCK}_{\mathsf{kr}}$/Gap-CDH, $\mathsf{eCK}_{\mathsf{nkr}}$/CDH	yes
Nets [38]	$\mathsf{eCK}_{\mathsf{kr}}$/Gap-CDH	$\mathsf{eCK}_{\mathsf{kr}}$/Gap-CDH, $\mathsf{eCK}_{\mathsf{nkr}}$/CDH	yes
Naxos+ [39]	$\mathsf{eCK}_{\mathsf{kr}}$/Gap-CDH	$\mathsf{eCK}_{\mathsf{kr}}$/CDH	yes
HMQV [31]*	$\mathrm{CK}_{\mathrm{HMQV}}$/Gap-CDH+KEA1	$\mathsf{eCK}_{\mathsf{kr}}$/Gap-CDH	no

Fig. 1. Obtained proofs for Key Exchange Protocols (*see explanation, nt=non-tight).

To compare different models of key distribution, we support two versions of the eCK model: The $\mathsf{eCK}_{\mathsf{nkr}}$ model where all static keys are honestly generated and the $\mathsf{eCK}_{\mathsf{kr}}$ model that allows the adversary to adaptively register arbitrary public keys for dishonest parties without providing a proof of possession. The original eCK model [36] sits in between our two versions. The adversary can register arbitrary public keys for dishonest parties *before* activating the first session, i.e., the registered public keys can depend on public keys of honest parties, but not on protocol messages, as, for example, required for Kaliski's attack [28] on MQV.

Our proof improves [33] in several ways: it uses the stronger eCK adversary model (with and without adversarial key registration); it supports proofs under standard assumptions (whereas the proof from [33] requires Gap assumptions), and; it exploits the Naxos trick resulting in simpler proof obligations for protocols that use it.

Concrete Proofs. We instantiate the generic proof to obtain security proofs for existing protocols; in all cases, the proofs of the simplified games are short by the standards of machine-checked proofs. Our results are summarized in Figure 1. Concretely, we prove that:

- Naxos and Nets are secure in the eCK model under the CDH assumption if keys are honestly generated. If we allow arbitrary adversarial key registration, we require the Gap-CDH assumption as in the original proof.
- The Naxos variant Naxos+ [39] is secure in the eCK model with arbitrary adversarial key registration under the CDH assumption. Here we obtain a similar result to the original proof using our generic proof method.
- A version of HMQV is secure in the eCK model under the Gap-CDH assumption. The version we analyze includes the identities and exchanged message in the input of the key derivation hash. The proof does not need KEA1 (knowledge of exponent assumption).

EasyCrypt Formalization. We have formalized all models, our generic proof for protocols using the Naxos trick, and the proofs for Nets, Naxos, and Naxos+ in EasyCrypt. Our formalization constitutes the biggest case study developed with the tool so far; e.g. the generic proof for protocols using the Naxos trick takes about 30,000 lines of code, including game definitions (about 50 of them), specifications, and proofs. On the other hand, the instantiation of the proof for concrete protocols is short and takes less than 1,000 lines each. Our

formalization also includes several reusable libraries that deal with random oracles, Twin DH, and common proof techniques such as plug and pray, that we discuss in Section 2.3.

Future Work

There are several directions for future work, including:

1. Automation and synthesis. The next logical step of this work is to *extend* our library of high-level principles to reason about AKE proofs in the random oracle model and provide more automation to simplify their application. These high-level principles will serve as a useful basis for future formalizations in EasyCrypt (beyond AKE), but will also make it faster to extend the current proof to support new features. They could also serve as a basis for fully automated proof methods and allow for the use of synthesis to generate secure protocols, following [3].

2. Extensions. We plan to strengthen our results in different directions. Possible extensions include adversary models with a more precise model of the CA [16], adversary models that allow reveal of different parts of state, and models of weak randomness. Moreover, we are also interested in using our framework to analyze larger protocols that use AKE as a subprotocol. This will be valuable to evaluate existing [45] and future proposals for secure transport-layer protocols.

3. Implementations. Our model provides a precise specification of the protocol. Using the techniques from [2], we intend to carry the security proof to executable implementations.

Related work

There is a vast body of literature on key-exchange protocols and on their associated security models; for a comparison between some existing models we refer to [34,20,22]. In addition to Naxos+, which we already mentioned, there are other protocols that achieve eCK-security under the CDH assumption, e.g., by Kim, Fujioka, and Ustaoglu [29] or by Pan and Wang [44].

There has been extensive work on the formal verification of key exchange protocols, see for instance the recent survey [15]. A significant amount of work is carried in the symbolic model, a high-level model which idealizes the treatment of cryptographic primitives. This level of abstraction makes the symbolic model amenable to automated analysis, and many tools have been developed for proving protocol security. Recent results focusing on DH-based key exchange protocols include [7], [35] and [46]. Over the last decade, many results on computational soundness results [1,21] have shown that under certain conditions, protocols deemed secure in the symbolic model remain secure in the computational model. Another related line of research (see, e.g., [12]) deals with the verification of implementations of security protocols such as TLS in the computational model.

2 Background

In this section, we give some background on notation, authenticated key exchange protocols, and EasyCrypt.

2.1 Notation

A^* denotes the set of all sequences with elements taken from A. For two sequences s_1 and s_2, we use $S_1 +\!\!+ S_2$ to denote their concatenation. We use $s_1 \xleftarrow{+\!\!+} s_2$ as a shorthand for the assignment $s_1 \leftarrow s_1 +\!\!+ s_2$. In the special case of bitstrings b_1 and b_2, we also use $b_1 \,\|\, b_2$ to denote their concatenation.

We use $A \rightharpoonup B$ to denote the set of partial functions from A to B. If f is a (partial) function, we define $f[a := b]$ as the function $x \mapsto \mathit{if}\ x = a\ \mathit{then}\ b\ \mathit{else}\ f(x)$. In games, we use $f[a] \leftarrow b$ as a shorthand for $f \leftarrow f[a := b]$ (update f at key a). For a finite set A, we use $x \xleftarrow{\$} A$ to denote that x is uniformly sampled from A.

We use $\mathbb{G}$ to denote a cyclic group of prime order p with generator g. We use $\mathbb{F}_p$ to denote the field of integers modulo p. We use $\mathsf{dlog}(Y)$ to denote the discrete logarithm of Y with respect to the basis g. We define $\mathsf{dh}(X, Y) \doteq X^{\mathsf{dlog}(Y)}$ and $\mathsf{ddh}(X, Y, Z) \doteq (\mathsf{dh}(X, Y) = Z)$. Based on the previous definitions, we define the following cryptographic assumptions. The challenger for DLOG gives $X \xleftarrow{\$} \mathbb{G}$ to the adversary who must return $\mathsf{dlog}(X)$. The challenger for CDH gives $X, Y \xleftarrow{\$} \mathbb{G}$ to the adversary who must return $\mathsf{dh}(X, Y)$. For SCDH, the adversary is given the same challenge, but must return a set containing $\mathsf{dh}(X, Y)$. We also define Gap versions [43] of these assumptions where the adversary is given access to an oracle that returns $\mathsf{ddh}(X, Y, Z)$ for arbitrary $X, Y, Z \in \mathbb{G}$.

2.2 One-Round Authenticated Key Exchange Protocols

In the following, we focus on one-round key exchange protocols. We believe most of our results can be extended to a more general notion of protocol. Further note that our results are not restricted to DH-based protocols and the formal definitions in Section 3.1 will generalize some of the notions we introduce informally in this section.

Figure 2 shows the computations and exchanged messages for a typical DH-based key exchange protocol. We assume a protocol consists of three components. First, there is a protocol component for key generation, which we show in the first line. Here, a participant $\hat{A}$ samples the *static secret key* a and computes the corresponding *static public key* A. Second, there is a component responsible for the distribution of the static public keys. We ignore the details for now and just assume that an agent can obtain the public key of another agent.

Finally, there is a component responsible for establishing the session key. This component consists of an *initiator role* and a *responder role*. If an agent $\hat{A}$ executes an instance of the initiator (resp. responder) role with the goal of establishing a session key with $\hat{B}$, we call this execution a session with *role* initiator (resp. responder), *actor* $\hat{A}$, and *peer* $\hat{B}$.

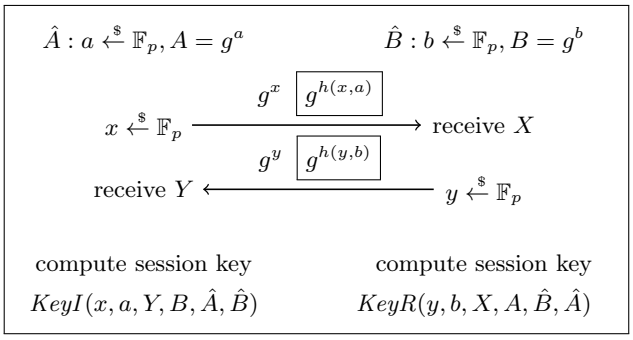

Fig. 2. Generic two pass protocol. Protocols using the Naxos trick use boxed messages.

When the initiator role is activated with actor $\hat{A}$ and peer $\hat{B}$, it first generates an *ephemeral secret key* x, computes the *ephemeral public key* X, sends it to $\hat{B}$, and waits for a reply. When the responder $\hat{B}$ is activated with a received message from $\hat{A}$, he stores the received message as X, generates y and Y in the same way as the initiator, sends Y to $\hat{A}$, and computes the *session key* using the $KeyR$ function. When the initiator is activated with the received message, he computes the session key using $KeyI$.

We can define the HMQV protocol by using $KeyI/KeyR$ that compute the key as $H(\sigma)$ for $\sigma = \mathsf{dh}(X, Y)\,\mathsf{dh}(X, B)^e\,\mathsf{dh}(A, Y)^d\,\mathsf{dh}(A, B)^{de}$, $e = \bar{h}(X, \hat{B})$, and $d = \bar{h}(Y, \hat{A})$. We can define the Naxos and Nets protocols by using the boxed expressions from Figure 2 to compute X and Y. These protocols both utilize the Naxos trick which combines the static and the ephemeral secret using the hash function h to obtain the exponent of ephemeral public key. Since the hash output is never stored and recomputed when required, these protocols are analyzed with respect to possible leakage of x or a, but leakage of $h(x, a)$ is not considered. The Naxos protocol defines the session key as $H(\sigma)$ for $\sigma = \mathsf{dh}(A, Y) \,\|\, \mathsf{dh}(X, B) \,\|\, \mathsf{dh}(X, Y) \,\|\, \hat{A} \,\|\, \hat{B}$. The Nets protocol defines the session key as $H(\sigma)$ for $\sigma = \mathsf{dh}(X, Y)\,\mathsf{dh}(X, B)\,\mathsf{dh}(A, Y)\,\mathsf{dh}(A, B) \parallel \mathsf{dh}(X, Y) \parallel \hat{A} \parallel \hat{B} \parallel X \parallel Y$. The Naxos+ protocol extends the Naxos protocols with the additional group element $\mathsf{dh}(A, B)$, i.e., the session key is defined as $H(\sigma)$ for $\sigma = \mathsf{dh}(A, Y) \,\|\, \mathsf{dh}(X, B) \,\|\, \mathsf{dh}(X, Y) \,\|\, \mathsf{dh}(A, B) \,\|\, \hat{A} \,\|\, \hat{B}$.

Informally, the security notion expected of such protocols is the following. If $\hat{A}$ completes a session with (honest) peer $\hat{B}$, then the session string computed by $\hat{A}$ is indistinguishable from a random bitstring for everyone except $\hat{B}$. It has been shown by Canetti and Krawczyk [18] that this is sufficient to establish a secure channel between $\hat{A}$ and $\hat{B}$. The secure channel can then be used for key confirmation. Recent adversary models like eCK build on this definition, but also allow for many scenarios where the adversary learns additional information such as ephemeral secret keys, static secret keys, or session keys. The eCK-model guarantees resilience against Unknown Key Share Attacks, Key Compromise Impersonation Resilience, and Weak Perfect Forward Secrecy, which we discuss at the end of Section 3.1.

2.3 EasyCrypt

EasyCrypt [6,5] is a machine-checked framework for building and verifying security proofs of cryptographic constructions. EasyCrypt follows the code-based, game-based approach to reductionist arguments: a proof consists of a series of probabilistic programs with adversarial code, called games, and of probabilistic claims relating the probability of one or more events in one or more games. However, EasyCrypt adopts a foundational approach, meaning that probabilistic claims, and the overall security statement, must all be justified to the last detail by means of elementary rules. Leveraging the state of the art in program verification, all probabilistic claims are proved using a probabilistic Relational Hoare Logic (pRHL), which generalizes Relational Hoare Logic [11] to a probabilistic setting. pRHL is a program logic whose judgments are of the form

$$\{\Phi\} c_1 \sim c_2 \{\Psi\}$$

where c_1 and c_2 are games and Φ and Ψ are relations on program states. The rules of pRHL allow to derive valid judgments, where a judgment as above is valid if for every initial memories m_1 and m_2 that are related by Φ, the output sub-distributions obtained by executing c_1 on m_1 and c_2 on m_2 respectively are related by $\Psi^{\#}$, where $\#$ is an operator that lifts relations on states to relations on sub-distributions over states. The definition of $\#$ is inspired from probabilistic process algebra. For suitable choices of Ψ, this implies inequalities of the form

$$\Pr[c_1, m_1 : E_1] \leq \Pr[c_2, m_2 : E_2]$$

which are typical in game-based proofs, i.e., the probability of event E_1 after executing c_1 in initial memory m_1 is upper-bounded by the probability of event E_2 after executing c_2 in m_2.

Although pRHL captures common patterns of reasoning in cryptographic proofs, there is an impedance mismatch between cryptographic practice and proofs built using pRHL; in particular, pRHL lacks mechanisms to instantiate previous results, and to apply high-level principles in proofs. To make matters precise, consider for instance the reduction of SCDH to CDH: using pRHL, one can prove that any instance of SCDH can be reduced to CDH, but one cannot perform the proof once and for all, and reuse the result. Fortunately, EasyCrypt now features a module system; the module system combines the power of module systems, as they exist in functional programming languages, with a system of capabilities that is used to restrict access to oracles or fragments of memories, as required in cryptography. The module system can be used for performing successive reductions locally, as often featured in pen-and-paper proofs. Module systems are essential to formalize complex proofs such as the ones considered here; indeed, previous attempts to carry out the generic proof without the module system were unsuccessful, because the adversary was carried explicitly throughout the proof, making reasoning unwieldy.

Additionally, the module system allows to prove general principles once and for all, and to carry out proofs simply by applying high-level principles. In our

formalization underlying this paper, we make extensive use of the following principles:

- lazy and eager sampling: this is used to switch back and forth between an implementation of a random function in which images are sampled on demand (lazily), or during initialization of the game (eagerly);
- plug and pray: if some event Φ happens for some $0 \leq i < n$, randomly sample a value j in this range and consider the event $\Phi \wedge i = j$ instead of Φ; and
- adversary prescience: this is used to provide an upper bound to the probability that an adversary guesses an unused value in the range of a random function

3 Model and Generic Proof

In this section, we first introduce our generic protocol model and our versions of the eCK model with and without adversarial key registration. Afterwards, we present our generic proof for protocols that employ the Naxos trick.

3.1 $\mathsf{eCK}_{\mathsf{kr}}$ Security and $\mathsf{eCK}_{\mathsf{nkr}}$ Security

We assume given a set ID of agent identities. We also define the set $\mathsf{Role} = \{\mathcal{I}, \mathcal{R}\}$ and the function $(\cdot)^\star : \mathsf{Role} \to \mathsf{Role}$ such that $\mathcal{I}^\star = \mathcal{R}$ and $\mathcal{R}^\star = \mathcal{I}$.

Generic Protocol Model. A protocol definition consists of instantiations for the types and functions given in the first column of Figure 3. These types and functions are instantiated as follows:

- The sequence $H_1 : \mathsf{I}_1 \to \mathsf{O}_1, \ldots, H_k : \mathsf{I}_k \to \mathsf{O}_k$ defines the types of hash functions used by the protocol.
- Sk defines the type of static secret keys, Pk defines the type of static public keys, Esk defines the type of ephemeral secret keys, Epk defines the type of ephemeral public keys, and Key defines the type of session keys.
- The function Pk defines how the static public key is computed from the static secret key and the function Epk defines how the ephemeral public key is computed from the ephemeral secret key and the static secret key.
- The functions $KeyI$ and $KeyR$ define how the session key is computed from the actor's secret data, the peer's public data, and the participants' identities. We use partial functions to capture failure, e.g., if a subgroup element check fails for one of the arguments.

We keep the distributions according to which the static and ephemeral secret keys are sampled implicit and assume they are uniformly sampled unless otherwise stated. The functions Epk, $KeyI$, and $KeyR$ can use the hash functions H_i. See Figure 3 for the Naxos^{core} instantiation of the generic model. In the next section, we will demonstrate how Naxos^{core} can be transformed into Naxos.

Types/Functions	Naxos^{core}	$\mathsf{Naxos} = \mathcal{T}^{\mathsf{hsk}}(\mathcal{T}^{\mathsf{nt}}(\mathsf{Naxos}^{core}))$
Hash functions	$\emptyset$	$H\colon \mathbb{G}^3 \times \mathsf{ID}^2 \to \{0,1\}^l$, $h\colon \mathbb{F}_p^2 \to \mathbb{F}_p$
Sk, Pk, Esk, Epk, Key	$\mathbb{F}_p$, $\mathbb{G}$, $\mathbb{F}_p$, $\mathbb{G}$, $\mathbb{G}^3 \times \mathsf{ID}^2$	$\mathbb{F}_p$, $\mathbb{G}$, $\mathbb{F}_p$, $\mathbb{G}$, $\{0,1\}^l$
$Pk\colon \mathsf{Sk} \to \mathsf{Pk}$	$a \mapsto g^a$	$a \mapsto g^a$
$Epk\colon \mathsf{Esk} \times \mathsf{Sk} \to \mathsf{Epk}$	$(x, \_) \mapsto g^x$	$(x, a) \mapsto g^{h(x,a)}$
$KeyI\colon \mathsf{Esk} \times \mathsf{Sk} \times \mathsf{Epk} \times \mathsf{Pk} \times \mathsf{ID} \times \mathsf{ID} \to \mathsf{Key}_\perp$	$(x, a, Y, B, \hat{A}, \hat{B}) \mapsto Y^a \,\|\, B^x \,\|\, Y^x \,\|\, \hat{A} \,\|\, \hat{B}$	$(x, a, Y, B, \hat{A}, \hat{B}) \mapsto H(Y^a \,\|\, B^{h(x,a)} \,\|\, Y^{h(x,a)} \,\|\, \hat{A} \,\|\, \hat{B})$
$KeyR\colon \mathsf{Esk} \times \mathsf{Sk} \times \mathsf{Epk} \times \mathsf{Pk} \times \mathsf{ID} \times \mathsf{ID} \to \mathsf{Key}_\perp$	$(y, b, X, A, \hat{B}, \hat{A}) \mapsto A^y \,\|\, X^b \,\|\, X^y \,\|\, \hat{A} \,\|\, \hat{B}$	$(x, a, Y, B, \hat{B}, \hat{A}) \mapsto H(A^{h(y,b)} \,\|\, X^b \,\|\, X^{h(y,b)} \,\|\, \hat{A} \,\|\, \hat{B})$

Fig. 3. Generic Protocol Model with Naxos^{core} instantiation and transformation.

Protocol Transformations. We define two transformations $\mathcal{T}^{\mathsf{hsk}}$ and $\mathcal{T}^{\mathsf{nt}}$. The first transformation $\mathcal{T}^{\mathsf{hsk}}$ modifies a protocol to hash the session key. The second transformation $\mathcal{T}^{\mathsf{nt}}$ modifies a protocol to utilize the Naxos trick. Figure 3 demonstrates how the Naxos protocol can be obtained by applying the two transformations to Naxos^{core}. We assume $\mathcal{T}^{\mathsf{hsk}}$ is implicitly parameterized by a positive integer l defining the size of the hash function output.

Given a protocol Π using hash functions $\boldsymbol{H}$, defining types Sk, Pk, Esk, Epk, Key, and defining functions Pk, Epk, $KeyI$, $KeyR$, the transformed protocols $\mathcal{T}^{\mathsf{hsk}}(\Pi)$ and $\mathcal{T}^{\mathsf{nt}}(\Pi)$ are defined as follows. We obtain $\mathcal{T}^{\mathsf{hsk}}(\Pi)$ from Π by adding a hash function $H : \mathsf{Key} \to \{0,1\}^l$ to $\boldsymbol{H}$, changing the type Key to $\{0,1\}^l$, redefining $KeyI$ in terms of the original $KeyI$ as $ki \mapsto H(KeyI(ki))$, and redefining $KeyR$ analogously. We obtain $\mathcal{T}^{\mathsf{nt}}(\Pi)$ from Π by adding a hash function $h : \mathsf{Esk} \times \mathsf{Sk} \to \mathsf{Esk}$ to $\boldsymbol{H}$ and redefining Epk in terms of the original Epk as $(x, a) \mapsto Epk(h(x, a), a)$. We also redefine the key computation to use $h(x, a)$ instead of x. Note that the original Epk usually ignores its second input and a is therefore only used as input to h in the computation of the ephemeral public key. We denote the composition of $\mathcal{T}^{\mathsf{nt}}$ and $\mathcal{T}^{\mathsf{hsk}}$ with $\mathcal{T}^{\mathsf{nt,hsk}}$.

Security Experiments. To define the games $\mathsf{eCK}_{\mathsf{kr}}$ and $\mathsf{eCK}_{\mathsf{nkr}}$ (with and without adversarial key registration), we first define the type St for the state of protocol sessions and the type Ev for the events required to express the security definition. We define St as $\mathsf{Role} \times \mathsf{Esk} \times \mathsf{Epk} \times \mathsf{ID} \times \mathsf{ID} \times \mathsf{Epk}_\perp \times \mathsf{Key}_\perp$. We define Ev as the data type generated by the constructors

$$\mathsf{EphRev} : \mathsf{Epk} \to \mathsf{Ev},\ \mathsf{KeyRev} : \mathsf{Epk} \to \mathsf{Ev}, \mathsf{StaticRev} : \mathsf{ID} \to \mathsf{Ev},$$
$$\mathsf{Accept} : \mathsf{Role} \times \mathsf{Epk} \times \mathsf{ID} \times \mathsf{ID} \times \mathsf{Epk} \to \mathsf{Ev}, \text{ and } \mathsf{Dishonest} : \mathsf{ID} \to \mathsf{Ev},$$

The main procedure of the games $\mathsf{eCK}_{\mathsf{kr},\Pi}(\mathcal{A})$ and $\mathsf{eCK}_{\mathsf{nkr},\Pi}(\mathcal{A})$ is given in the first column of Figure 4. We assume that the adversary $\mathcal{A}$ consists of the two procedures $\mathcal{A}_1$ and $\mathcal{A}_2$ sharing state. In the games, the adversary is provided

with access to the oracles defined in the second column of Figure 4 and with random oracle access using wrappers $H_1^{\mathcal{A}}, \ldots, H_k^{\mathcal{A}}$. The oracles *establishHonest* and *establishDishonest* (only in $\mathsf{eCK}_{\mathsf{kr},\Pi}$) allow the adversary to establish honest agents for which the keys are sampled and dishonest agents where the public key can be chosen. For honest agents, the adversary can control the execution of initiator and responder sessions using $init_1$, $init_2$, and *resp*. Dishonest agents can be used as peers of protocol sessions, but cannot be used as actors since the static secret key is required to execute the protocol. The remaining oracles allow the adversary to reveal static secrets, ephemeral secrets, and session keys.

The adversary wins if he can distinguish the session key of the test session from a random session key and the test session is fresh, i.e., he did not perform forbidden reveal queries. The freshness condition is formalized using the *fresh* predicate given in Figure 5. We use the ephemeral public key to identify a session for session key reveals and ephemeral reveals.

Discussion. In $\mathsf{eCK}_{\mathsf{kr}}$, we allow the actor of the test session to execute sessions with dishonest users, but the actor and peer of the test session itself must be honest. In both $\mathsf{eCK}_{\mathsf{kr}}$ and $\mathsf{eCK}_{\mathsf{nkr}}$, we disallow $\hat{A} = \hat{B}$ because many deployed protocols disallow this case or use distinct keys for different roles. It would be possible to lift this limitation at the cost of additional proof obligations for users of the generic proof.

The freshness condition captures Unknown Key Share Attacks because if $\hat{A}$ establishes a key with $\hat{B}$, but $\hat{B}$ believes that he shares this key with $\hat{C} \neq \hat{A}$, then there are two non-matching sessions with the same session key and one of them can be revealed. It captures Key Compromise Impersonation because leakage of the actors static secret key is allowed for the test session. It also captures Weak Perfect Forward Secrecy because for all sessions where the adversary is passive (there is a matching session), reveals for all ephemeral secrets, except for those of the test session and its matching session, and for all static secrets are allowed. The stronger notion of Perfect Forward Secrecy requires changes to the freshness condition and we leave such an extension of our results open for future work.

In our definition of Naxos, we use the type $\mathbb{G}$ for ephemeral and static public keys. This models an implementation ensuring that these values are elements of $\mathbb{G}$. It is also possible to use a "larger type" and explicitly model the required checks using failure in the key computation functions.

3.2 Generic Proof

Before presenting our generic proof, we define three properties of core protocols:

(P1) The functions *Pk* and *Epk* are injective.
(P2) $KeyI(x, a, Y, B, \hat{A}, \hat{B})$ is efficiently computable from $KeyR(x, a, Y, B, \hat{A}, \hat{B})$.
(P3) If two distinct sessions $(X, Y, \hat{A}, \hat{B}, r)$ and $(X', Y', \hat{A}', \hat{B}', r')$ compute the same session key, then $(X, Y, \hat{A}, \hat{B}) = (Y', X', \hat{B}', \hat{A}')$ and $\{r, r'\} = \{\mathcal{I}, \mathcal{R}\}$.

Game:	Oracles ($\mathsf{eCK}_{\mathsf{nkr}}$ does not include $\mathit{establishDishonest}$):
var $evs : \mathsf{Ev}^* = [\,]$ var $ses : \mathsf{Nat} \rightharpoonup \mathsf{St} = \emptyset$ var $sks : \mathsf{ID} \rightharpoonup \mathsf{Sk} = \emptyset$ var $pks : \mathsf{ID} \rightharpoonup \mathsf{Pk} = \emptyset$ var $i : \mathsf{Nat} = 0$ $t \leftarrow \mathcal{A}_1()$ $(r, \_, X, \hat{A}, \hat{B}, Y, k)$ $\quad \leftarrow ses[t]$ $b \xleftarrow{\$} \{0,1\}$ $k' \xleftarrow{\$} \mathsf{Key}$ $b' \leftarrow \mathcal{A}_2(b \,?\, k : k')$ $t \leftarrow (r, X, \hat{A}, \hat{B}, Y)$ return $\quad b = b' \wedge \mathit{fresh}_{evs}(t)$	$\mathit{init}_1(\hat{A}, \hat{B}) : \mathsf{Epk} =$ $\quad i \leftarrow i + 1$ $\quad a \leftarrow sks[\hat{A}]$ $\quad$ if $a = \bot \vee \hat{A} = \hat{B}$ then $\quad\quad$ return $\bot$ $\quad x \xleftarrow{\$} \mathsf{Esk};\ X \leftarrow Epk(x, a)$ $\quad ses[i] \leftarrow (\mathcal{I}, x, X, \hat{A}, \hat{B}, \bot, \bot)$ $\quad$ return (i, X) $\mathit{init}_2(i, Y) =$ $\quad (\mathcal{I}, x, X, \hat{A}, \hat{B}, \bar{Y}, \_) \leftarrow ses[i]$ $\quad$ if $\bar{Y} \neq \bot$ then return $\bot$ $\quad a \leftarrow sks[\hat{A}];\ B \leftarrow pks[\hat{B}]$ $\quad k \leftarrow KeyI(x, a, Y, B, \hat{A}, \hat{B})$ $\quad ses[i] \leftarrow (\mathcal{I}, x, X, \hat{A}, \hat{B}, Y, k)$ $\quad$ if $k = \bot$ then return $\bot$ $\quad evs \xleftarrow{+\!\!+} \mathsf{Accept}(\mathcal{I}, X, \hat{A}, \hat{B}, Y)$ $\mathit{resp}(\hat{B}, \hat{A}, X) : \mathsf{Epk} =$ $\quad i \leftarrow i + 1$ $\quad$ if $\hat{A} = \hat{B}$ then return $\bot$ $\quad b \leftarrow sks[\hat{B}];\ A \leftarrow pks[\hat{A}]$ $\quad y \xleftarrow{\$} \mathsf{Esk};\ Y \leftarrow Epk(y, b)$ $\quad k \leftarrow KeyR(y, b, X, A, \hat{B}, \hat{A})$ $\quad$ if $k = \bot$ then return $\bot$ $\quad ses[i] \leftarrow (\mathcal{R}, y, Y, \hat{B}, \hat{A}, X, k)$ $\quad evs \xleftarrow{+\!\!+} \mathsf{Accept}(\mathcal{R}, Y, \hat{B}, \hat{A}, X)$ $\quad$ return (i, Y)	$\mathit{establishHonest}(\hat{A}) : \mathsf{Pk} =$ $\quad$ if $pks[\hat{A}] \neq \bot$ then $\quad\quad$ return $\bot$ $\quad sks[\hat{A}] \xleftarrow{\$} \mathsf{Sk}$ $\quad pks[\hat{A}] \leftarrow Pk(sks[\hat{A}])$ $\quad$ return $pks[\hat{A}]$ $\mathit{establishDishonest}(\hat{A}, A) =$ $\quad$ if $pks[\hat{A}] \neq \bot$ then $\quad\quad$ return $\bot$ $\quad pks[\hat{A}] \leftarrow A$ $\quad evs \xleftarrow{+\!\!+} \mathsf{Dishonest}(\hat{A})$ $\mathit{staticRev}(\hat{A}) : \mathsf{Sk} =$ $\quad evs \xleftarrow{+\!\!+} \mathsf{StaticRev}(\hat{A})$ $\quad$ return $sks[\hat{A}]$ $\mathit{ephRev}(i) : \mathsf{Esk} =$ $\quad (\_, x, X, \_, \_, \_, \_) \leftarrow ses[i]$ $\quad evs \xleftarrow{+\!\!+} \mathsf{EphRev}(X)$ $\quad$ return x $\mathit{keyRev}(i) : \mathsf{Key} =$ $\quad (\_, \_, X, \_, \_, \_, k) \leftarrow ses[i]$ $\quad evs \xleftarrow{+\!\!+} \mathsf{KeyRev}(X)$ $\quad$ return k

Fig. 4. Games $\mathsf{eCK}_{\mathsf{kr},\Pi}(\mathcal{A})$ and $\mathsf{eCK}_{\mathsf{nkr},\Pi}(\mathcal{A})$ for $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and protocol Π.

We assume the second property for simplicity. For core protocols, which we consider here, it usually suffices to reorder the key string elements to obtain $KeyI(ki)$ from $KeyR(ki)$. The third property is called strong partnering in [33] and ensures key independence.

Exploiting the Naxos Technique. We exploit that for protocols $\mathcal{T}^{\mathsf{nt}}(\Pi)$ using the Naxos technique, both x and a are required to learn the secret input $h(x, a)$ of Epk. This is a consequence of the fact that the value $h(x, a)$ cannot be revealed by the adversary in the eCK model. This decision is motivated by the assumption that honest agents executing the protocol never store the value $h(x, a)$. We can therefore prove security of Π in a restricted $\mathsf{eCK}^{\mathsf{nt}}$ game to obtain eCK-security of $\mathcal{T}^{\mathsf{nt}}(\Pi)$. For $m \in \{\mathsf{kr}, \mathsf{nkr}\}$, we obtain $\mathsf{eCK}^{\mathsf{nt}}_m$ from eCK_m by replacing ephRev

$fresh_{evs}(r, X, \hat{A}, \hat{B}, Y) \doteq$

There is no session key reveal for t, not both ephemeral and static reveal for t,
$\mathsf{KeyRev}(X) \notin evs \wedge \neg(\mathsf{EphRev}(X) \in evs \wedge \mathsf{StaticRev}(\hat{A}) \in evs)$
and the adversary did not register $\hat{A}$'s or $\hat{B}$'s public keys.

$\wedge\ \mathsf{Dishonest}(\hat{A}) \notin evs \wedge \mathsf{Dishonest}(\hat{B}) \notin evs$

If there is a matching session t', then

$\wedge\ (\mathsf{Accept}(r^\star, Y, \hat{B}, \hat{A}, X) \in evs \implies$
there is no key reveal for t' and not both ephemeral and static reveal for t'.
$(\mathsf{KeyRev}(Y) \notin evs \wedge \neg(\mathsf{EphRev}(Y) \in evs \wedge \mathsf{StaticRev}(\hat{B}) \in evs)))$

If there is no matching session, then there is no static reveal for B.

$\wedge\ (\mathsf{Accept}(r^\star, Y, \hat{B}, \hat{A}, X) \notin evs \implies \mathsf{StaticRev}(\hat{B}) \notin evs)$

Fig. 5. Freshness condition for a trace evs and a test session $t = (r, X, \hat{A}, \hat{B}, Y)$.

with $ephRev^{\mathsf{nt}}$ as defined below:

$$
\begin{array}{l}
ephRev^{\mathsf{nt}}(i, a) : \mathsf{Esk} = \\
\quad (\_, x, X, \hat{A}, \_, \_, \_) \leftarrow ses[i] \\
\quad \mathsf{if}\ a \neq sks[\hat{A}]\ \mathsf{then\ return}\ \bot \\
\quad evs \xleftarrow{+\!\!+} \mathsf{EphRev}(X) \\
\quad \mathsf{return}\ x
\end{array}
$$

Informally, our reduction exploits that x for Π in $\mathsf{eCK}^{\mathsf{nt}}$ corresponds to $h(x, a)$ for $\mathcal{T}^{\mathsf{nt}}(\Pi)$ in eCK and a query $ephRev^{\mathsf{nt}}(i, a)$ in $\mathsf{eCK}^{\mathsf{nt}}$ corresponds to the sequence of queries $x \leftarrow ephRev(i)$; $h^{\mathcal{A}}(x, a)$ in eCK.

To state our lemma, we define $\mathcal{A}$ to be a $(q_{se}, q_{ag}, q_{\boldsymbol{H}})$ eCK_m (or $\mathsf{eCK}^{\mathsf{nt}}_m$) adversary if $\mathcal{A}$ activates at most q_{se} sessions involving at most q_{ag} agents and performs at most q_{H_i} queries to the random oracle $H_i^{\mathcal{A}}$. We use q_h to denote the number of queries to the random oracle $h^{\mathcal{A}}$ introduced by the $\mathcal{T}^{\mathsf{nt}}$ transformation.

Lemma 1. *Let $m \in \{\mathsf{kr}, \mathsf{nkr}\}$, Π be a protocol, and $\mathcal{A}$ a $(q_{se}, q_{ag}, q_{\boldsymbol{H}})$ eCK_m adversary. Then there is a $(q_{se}, q_{ag}, q_{\boldsymbol{H}})$ $\mathsf{eCK}^{\mathsf{nt}}_m$ adversary $\mathcal{B}$ such that*

$$\Pr\left[\mathsf{eCK}_{m,\mathcal{T}^{\mathsf{nt}}(\Pi)}(\mathcal{A}) = 1\right] \leq \Pr\left[\mathsf{eCK}^{\mathsf{nt}}_{m,\Pi}(\mathcal{B}) = 1\right] + \epsilon_{\mathcal{T}^{\mathsf{nt}}}$$

where $\epsilon_{\mathcal{T}^{\mathsf{nt}}} = {}^{2\,q_h\,q_{se}}/_{|\mathsf{Esk}|} + {}^{q_{se}^2}/_{2\,|\mathsf{Esk}|}$. Furthermore, the adversary $\mathcal{B}$ runs in time at most $\mathcal{O}(q_h\, t_{Pk} + t_{\mathcal{A}})$ where t_{Pk} is the time required to compute Pk.

In our EasyCrypt formalization, we explicitly construct the simulator $\mathcal{S}$ sketched in the proof below and prove the probability statement for $\mathcal{B} = \mathcal{S}(\mathcal{A})$.

Proof (Sketch). After bounding the probability of collisions for ephemeral secrets and bounding the probability of the adversary querying $h^{\mathcal{A}}(x, *)$ for an ephemeral secret x before revealing it, we define a simulator $\mathcal{B}$ that calls $\mathcal{A}$ and

$G^{\mathsf{hsk,nt}}_{1,m}$ (a secret):	$G^{\mathsf{hsk,nt}}_{2,m}$ (x, b secret):	$G^{\mathsf{hsk,nt}}_{3}$ (x, y secret):
$a \xleftarrow{\$} \mathsf{Sk}$; $A \leftarrow Pk(a)$ $\boldsymbol{z} \xleftarrow{\$} \mathsf{Esk}^{q_{se}}$; $\boldsymbol{Z} \leftarrow Epk(\boldsymbol{z})$ $\boldsymbol{c} \xleftarrow{\$} \mathsf{Sk}^{q_{ag}-1}$ $a' \leftarrow \mathcal{B}_1(A, \boldsymbol{Z}, \boldsymbol{c})$ return $(a = a')$	$x \xleftarrow{\$} \mathsf{Esk}$; $X \leftarrow Epk(x)$ $b \xleftarrow{\$} \mathsf{Sk}$; $B \leftarrow Pk(b)$ $\boldsymbol{z} \xleftarrow{\$} \mathsf{Esk}^{q_{se}-1}$; $\boldsymbol{Z} \leftarrow Epk(\boldsymbol{z})$ $\boldsymbol{c} \xleftarrow{\$} \mathsf{Sk}^{q_{ag}-1}$ $(i, Y, \hat{A}, \hat{B}, S) \leftarrow \mathcal{B}_2(X, B, \boldsymbol{c}, \boldsymbol{Z})$ $k \leftarrow KeyI(x, c_i, Y, B, \hat{A}, \hat{B})$ return $(k \in S \wedge k \neq \bot)$	$x \xleftarrow{\$} \mathsf{Esk}$; $X \leftarrow Epk(x)$ $y \xleftarrow{\$} \mathsf{Esk}$; $Y \leftarrow Epk(y)$ $\boldsymbol{c} \xleftarrow{\$} \mathsf{Sk}^{q_{ag}}$ $(i, j, \hat{A}, \hat{B}, S) \leftarrow \mathcal{B}_3(X, Y, \boldsymbol{c})$ $C \leftarrow Pk(c_j)$ $k \leftarrow KeyI(x, c_i, Y, C, \hat{A}, \hat{B})$ return $(k \in S \wedge k \neq \bot)$
$eqS^{\mathsf{kr}}(i, Y, \underline{C}, \hat{A}, \hat{C}, k) =$ $ki \leftarrow (z_i, a, Y, \underline{C}, \hat{A}, \hat{C})$ return $(k = KeyI(ki))$	$eqS^{\mathsf{kr}}(j, W, \underline{C}, \hat{B}, \hat{C}, k) =$ $ki \leftarrow (z_j, b, W, \underline{C}, \hat{B}, \hat{C})$ return $(k = KeyI(ki))$	
$eqS^{\mathsf{nkr}}(i, Y, \underline{j}, \hat{A}, \hat{C}, k) =$ $\underline{C \leftarrow Pk(c_j)}$ $ki \leftarrow (z_i, a, Y, \underline{C}, \hat{A}, \hat{C})$ return $(k = KeyI(ki))$	$eqS^{\mathsf{nkr}}(j, W, \underline{u}, \hat{B}, \hat{C}, k) =$ $\underline{C = Pk(c_u)}$ $ki \leftarrow (z_j, b, W, \underline{C}, \hat{B}, \hat{C})$ return $(k = KeyI(ki))$	

Fig. 6. Games defining $\mathsf{CSK}^{\mathsf{nt}}_{\mathsf{kr},\Pi}(\mathcal{B})$ and $\mathsf{CSK}^{\mathsf{nt}}_{\mathsf{nkr},\Pi}(\mathcal{B})$ for $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ with alternative eqS-oracle definitions for kr and nkr.

handles queries as follows: On queries $init_1$ and $resp$, $\mathcal{B}$ updates a mapping from the session index i of the started session to the public key A_i of i's actor. For queries $ephRev(i)$, $\mathcal{B}$ samples and stores the value $\bar{x}_i$ ensuring that there are no collisions and that answers are consistent, i.e., $\mathcal{B}$ simulates the ephemeral secrets in $\mathsf{eCK}_{m,\mathcal{T}^{\mathsf{nt}}(\Pi)}$. On query $h^{\mathcal{A}}(z, c)$, $\mathcal{B}$ checks if there is an i such that $z = \bar{x}_i$ and $Pk(c) = A_i$ (which implies $a_i = c$, i.e., c is equal to the static secret key of the i-th session) and returns $ephRev^{\mathsf{nt}}(i, a_i)$ if the check succeeds and $h(z, c)$ otherwise. All other queries are just forwarded. In the reduction, the ephemeral secrets x_i in $\mathsf{eCK}^{\mathsf{nt}}_{m,\Pi}$ correspond to hash values $h(x_i, a_i)$ in $\mathsf{eCK}_{m,\mathcal{T}^{\mathsf{nt}}(\Pi)}$. □

Exploiting the Hashing of the Session Key. The $\mathsf{CSK}^{\mathsf{nt}}_{\mathsf{nkr}}$ and $\mathsf{CSK}^{\mathsf{nt}}_{\mathsf{kr}}$ models for protocols that employ the Naxos technique are defined by the three games given in Figure 6. The winning conditions state that the adversary must compute certain keys. They result from case distinctions where we show that the adversary cannot win unless he queries these keys to $ephRev^{\mathsf{nt}}$ or the random oracle H. We first describe the games and then explain how they are used in the reduction.

$G^{\mathsf{hsk,nt}}_{1,m}$**:** The adversary is given a static public key A, a vector $\boldsymbol{Z}$ of ephemeral public keys, and a vector $\boldsymbol{c}$ of static secret keys. To win, he must return the static secret key a for A. He is given access to a decision oracle that returns 1 if the given k is the session key for a session with session data $(z_i, a, , Y, C, \hat{A}, \hat{C})$ where z_i must be an element of $\boldsymbol{z}$, a is fixed, and $\hat{A}$, $\hat{C}$,

and Y can be arbitrary. For $m = \mathsf{kr}$, C can be arbitrary. For $m = \mathsf{nkr}$, C must be an element of $Pk(\boldsymbol{c})$ reflecting that keys are honestly generated.

$G_{2,m}^{\mathsf{hsk},\mathsf{nt}}$**:** The adversary is given an ephemeral public key X, a static public key B, a vector $\boldsymbol{Z}$ of ephemeral public keys, and a vector $\boldsymbol{c}$ of static secret keys. He chooses a static secret key c_i from $\boldsymbol{c}$, an arbitrary ephemeral public key Y, and arbitrary agent identities $\hat{A}$ and $\hat{B}$. To win, he must return a set S that contains the session key for $(x, c_i, Y, B, \hat{A}, \hat{B})$. He is provided with access to a decision oracle that returns 1 if the given k is the session key for a session with session data $(z_j, b, W, C, \hat{B}, \hat{C})$ where z_j must be an element of $\boldsymbol{z}$, b is fixed, and W, $\hat{B}$, and $\hat{C}$ can be arbitrary. For $m = \mathsf{kr}$, the static public key C of the peer can be arbitrary. For $m = \mathsf{nkr}$, C must be an element of $Pk(\boldsymbol{c})$.

$G_{3}^{\mathsf{hsk},\mathsf{nt}}$**:** The adversary is given ephemeral public keys X, Y and a vector $\boldsymbol{c}$ of static secret keys. He chooses static secret keys c_i, c_j in $\boldsymbol{c}$ and arbitrary $\hat{A}$, $\hat{B}$. To win, he must return a set S that contains the session key for $(x, c_i, Y, Pk(c_j), \hat{A}, \hat{B})$.

In the reduction, we use $G_{1,m}^{\mathsf{hsk},\mathsf{nt}}$ to handle the case where the adversary queries $ephRev^{\mathsf{nt}}(i, a)$ without revealing the static secret a for some $\hat{A}$. For the remaining cases, we know that the ephemeral secret x of the test session must be secret. We use $G_{2,m}^{\mathsf{hsk},\mathsf{nt}}$ to handle the case where the static secret b of the test session's peer remains unrevealed and $G_{3}^{\mathsf{hsk},\mathsf{nt}}$ to handle the case where b is revealed and there is a matching session with unrevealed ephemeral secret y. The *eqS* oracle in $G_{1,m}^{\mathsf{hsk},\mathsf{nt}}$ is used to synchronize queries to $H^{\mathcal{A}}$ and *keyRev* for $\hat{A}$'s sessions. Analogously, *eqS* in $G_{2}^{\mathsf{hsk},\mathsf{nt}}$ is used for $\hat{B}$'s sessions. We can now state our main theorem using q_h (resp. q_H) to denote the number of queries to the oracle introduced by $\mathcal{T}^{\mathsf{nt}}$ (resp. $\mathcal{T}^{\mathsf{hsk}}$).

Theorem 1 *Let $m \in \{\mathsf{kr}, \mathsf{nkr}\}$ and Π be a protocol satisfying properties* **P1**–**P3**. *Let $\mathcal{A}$ be a $(q_{se}, q_{ag}, q_{\boldsymbol{H}})$ eCK_m adversary. Then there are $\mathsf{CSK}_m^{\mathsf{nt}}$ adversaries $\mathcal{B}_1$–$\mathcal{B}_3$ such that*

$$\begin{aligned} &2\Pr\left[\mathsf{eCK}_{m,\mathcal{T}^{\mathsf{nt},\mathsf{hsk}}(\Pi)}(\mathcal{A}) = 1\right] - 1 \\ &\quad \leq \delta_1 \Pr\left[G_{1,m,\Pi}^{\mathsf{hsk},\mathsf{nt}}(\mathcal{B}_1) = 1\right] + \delta_2 \Pr\left[G_{2,m,\Pi}^{\mathsf{hsk},\mathsf{nt}}(\mathcal{B}_2) = 1\right] \\ &\quad + \delta_3 \Pr\left[G_{3,\Pi}^{\mathsf{hsk},\mathsf{nt}}(\mathcal{B}_3) = 1\right] + \epsilon_{\mathcal{T}^{\mathsf{nt},\mathsf{hsk}}} \end{aligned}$$

for $\epsilon_{\mathcal{T}^{\mathsf{nt},\mathsf{hsk}}} = {}^{(2\,q_h\,q_{se} + 2\,q_{se}^2)}/_{|\mathsf{Esk}|}$, $\delta_1 = q_{ag}$, $\delta_2 = q_{ag}\,q_{se}$, and $\delta_3 = q_{se}^2$. Furthermore, the adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ perform at most $q_H\,q_{se}$ queries to eqS and the adversaries $\mathcal{B}_2$ and $\mathcal{B}_3$ return sets of size at most $2\,q_H$. The adversaries $\mathcal{B}_1$–$\mathcal{B}_3$ run in time at most $\mathcal{O}((q_h + q_{ag})\,t_{Pk} + q_{se}\ t_{proto} + q_{se}\,q_H + t_{\mathcal{A}})$ where t_{proto} denotes the time to execute a protocol session.

Proof (Sketch). We first apply Lemma 1. Then it remains to prove that $\mathsf{CSK}_m^{\mathsf{nt}}$-security of Π implies $\mathsf{eCK}_m^{\mathsf{nt}}$-security of $\mathcal{T}^{\mathsf{hsk}}(\Pi)$. Let σ denote the input to H

Game:	Oracles:
var $evs : \mathsf{Ev}^* = [\,]$ var $sks : \mathsf{ID} \rightharpoonup \mathsf{Sk} = \emptyset$ var $pks : \mathsf{ID} \rightharpoonup \mathsf{Pk} = \emptyset$ var $ses : \mathsf{Nat} \rightharpoonup \mathsf{St} = \emptyset$ var $i : \mathsf{Nat} = 0$ $(S, t) \leftarrow \mathcal{A}_1()$ $(r, \_, X, \hat{A}, \hat{B}, Y, k) \leftarrow ses[t]$ $ts \leftarrow (r, X, \hat{A}, \hat{B}, Y)$ return $(k \in S \wedge k \neq \bot \wedge fresh_{evs}(ts))$	Replace *keyRev* and $h^{\mathcal{A}}$ with *eqS*. Keep other oracles. $eqS(i, k) : \mathsf{Key} =$ $(r, \_, X, \hat{A}, \hat{B}, Y, k') \leftarrow ses[i]$ $evs \leftarrow^{++} \mathsf{KeyRev}(r, X, \hat{A}, \hat{B}, Y)$ return $(k = k')$

Fig. 7. Intermediate game GI used in reductions ($\mathsf{eCK}^{\mathsf{nt}}$ to $\mathsf{CSK}^{\mathsf{nt}}$ and eCK to CSK).

used to compute the session key of the test session. We first bound the probability that the adversary wins without querying σ to $H_{\mathcal{A}}$ by $1/2$. First, note that he cannot reveal a session key with hash input σ since condition **P3** for Π implies that the corresponding session is either a matching session or the test session itself (up to collisions of ephemeral secrets and guessing of unused ephemeral secrets). He therefore receives a key that is sampled independently of his view for both values of b and cannot do better than guessing b in this case.

We now proceed by bounding the probability of $\sigma \in Q_H \wedge fresh_{evs}(sid)$ in $\mathsf{eCK}^{\mathsf{nt}}_{m,\mathcal{T}^{\mathsf{hsk}}(\Pi)}$ where $sid = (r, X, \hat{A}, \hat{B}, Y)$ and Q_H is the set of values queried to H by the adversary. Our goal is to perform a reduction to the intermediate game $GI_{m,\Pi}$ shown in Figure 7. The simulator will use the *eqS* oracle in $GI_{m,\Pi}$ to simulate the oracles $H^{\mathcal{A}}$ and *keyRev* and return (t, Q_H). The *eqS* oracle is used to synchronize values returned in *keyRev* and H, but it cannot be used for the call to H for σ in the main body. We therefore perform a sequence of steps that includes enforcing a (monotonous version of) freshness to remove this call before performing the reduction.

To obtain the three games $G^{\mathsf{hsk},\mathsf{nt}}_{1,m,\Pi}$, $G^{\mathsf{hsk},\mathsf{nt}}_{2,m,\Pi}$, and $G^{\mathsf{hsk},\mathsf{nt}}_{3,\Pi}$ from $GI_{m,\Pi}$, we perform two case distinctions followed by one reduction for each case. The first case distinction is for the event that the adversary queries $\mathsf{EphRev}(i, a)$ without performing $\mathsf{StaticRev}(\hat{A})$ and revealing a beforehand. To bound this probability, we first guess $\hat{A}$ and then perform a reduction to $G^{\mathsf{hsk},\mathsf{nt}}_{1,m,\Pi}$. Since the adversary can reveal all secrets except for a and the ephemeral secret keys of $\hat{A}$, the simulator receives the static secret keys $\boldsymbol{c}$ of the other agents, the ephemeral public keys of $\hat{A}$'s sessions, and A. The simulator samples all other values himself and can simulate all oracles on its own, except for *eqS* where the provided oracle is used for $\hat{A}$'s sessions. If $m = \mathsf{nkr}$, all keys are honestly generated and for all queries to *eqS*, the static public key of the peer is equal to $Pk(c)$ for some $c \in \boldsymbol{c}$. If $m = \mathsf{kr}$, the static public key of the peer can be arbitrary.

Before performing the second case distinction, we guess the test session. Since there is no ephemeral reveal without a previous static reveal, the test session's ephemeral secret x cannot be revealed. We now perform a case distinction if the

Game G_{2DDH}:	Game G:
$x \xleftarrow{\$} \mathbb{F}_p;\ X \leftarrow g^x$ $\boldsymbol{y} \xleftarrow{\$} \mathbb{F}_p^n;\ \boldsymbol{Y} \leftarrow g^{\boldsymbol{y}}$ $z \xleftarrow{\$} \mathbb{F}_p;\ Z \leftarrow g^z$ $t \leftarrow \mathcal{A}^{\mathsf{2DDH}}(X, \boldsymbol{Y}, Z)$ $\mathsf{return}\ \phi(X, Z, t)$ $\mathsf{2DDH}(i, \hat{Z}, U, V) = \mathsf{return}$ $\quad \mathsf{ddh}(X, \hat{Z}, U) \wedge \mathsf{ddh}(Y_i, \hat{Z}, U)$	$x \xleftarrow{\$} \mathbb{F}_p;\ X \leftarrow g^x$ $z \xleftarrow{\$} \mathbb{F}_p;\ Z \leftarrow g^z$ $t \leftarrow \mathcal{B}(X, Z)$ $\mathsf{return}\ \phi(X, Z, t)$

Fig. 8. Twin DDH games G_{2DDH} and G.

adversary reveals the static secret key b of the peer $\hat{B}$ of the test session. If not, then we know that x, b, and the ephemeral secret keys of $\hat{B}$'s sessions are secret. To perform the reduction to $G^{\mathsf{hsk,nt}}_{2,m,\Pi}$, we guess $\hat{B}$ and define a simulator that receives X, B, the static secret keys $\boldsymbol{c}$ of all agents except $\hat{B}$, and the ephemeral public keys of $\hat{B}$'s sessions. The simulator samples all other values himself and can simulate all oracles on its own, except for *eqS* where the provided oracle is used for $\hat{B}$'s sessions. Like in the previous case, the static public key of the peer is equal to $Pk(c)$ for some $c \in \boldsymbol{c}$ if $m = \mathsf{nkr}$ and arbitrary otherwise.

In the last case, there is a static reveal for the peer $\hat{B}$ of the test session. Hence, there must be a matching session with ephemeral secret key y and the only other value that cannot be revealed is x. We guess the matching session and since *eqS* queries for the test session and the matching session are forbidden, the simulator for $G^{\mathsf{hsk,nt}}_{3,\Pi}$ can simulate the *eqS* oracle on its own in this case. □

4 Trapdoor Test, Twin DH, and (S)CDH

To minimize the EasyCrypt proof effort, we first prove a generalized version of the Twin DH Assumption from [19]. We use this result for the protocol proofs and to obtain a tighter reduction from CDH to SCDH based on Shoup's self corrector [47].

Twin DH. In the original Twin DH assumption, the adversary is given challenges $X, Y, Z \in \mathbb{G}$ and has to compute the group elements $(\mathsf{dh}(X, Z), \mathsf{dh}(Y, Z))$ given oracle access to

$$\mathsf{2DDH}(\hat{Z}, U, V) \doteq (\mathsf{ddh}(X, \hat{Z}, U) \wedge \mathsf{ddh}(Y, \hat{Z}, V)).$$

The value Y is called the "twin" of X and the assumption can be seen as a "twin version" of the Strong DH assumption, which is a variant of Gap CDH where the first input of the DDH oracle is fixed. In contrast to these two assumptions, Twin DH follows from CDH in all groups since the 2DDH oracle can be simulated using the trapdoor test.

Our generalization uses n twins $Y_1, \ldots, Y_n$ of X instead of a single twin and consequently provides a $\mathsf{2DDH}$ oracle that can be used with all twins X, Y_i. Concretely, for a predicate ϕ, we define the two games G_{2DDH} and G given in Figure 8 and prove the following lemma for which the proof can be found in Appendix A.

Lemma 2. *Let $\mathcal{A}$ be a G_{2DDH} adversary that performs at most q queries to $\mathsf{2DDH}$. Then there exists a G adversary $\mathcal{B}$ such that*

$$\Pr\left[G_{\mathsf{2DDH}}(\mathcal{A}) = 1\right] \leq \Pr\left[G(\mathcal{B}) = 1\right] + q/p.$$

Moreover, $\mathcal{B}$ runs in time $\mathcal{O}(T_{\mathcal{A}} + q\, t_{\mathbb{G}} + n\, t_{\mathbb{G}})$ where $t_{\mathbb{G}}$ denotes the time required to perform a group operation such as exponentiation or division.

We define the following reductions as instantiations of this lemma:
- $\mathsf{CDH}_{\mathsf{2DDH}}$ to CDH for $\phi(X, Z, U) \doteq \mathsf{dh}(X, Z) = U$.
- $\mathsf{DLOG}_{\mathsf{2DDH}}$ to DLOG for $\phi(X, Z, x') \doteq X = g^{x'}$.
- $\mathsf{SCDH}_{\mathsf{2DDH}}$ to SCDH for $\phi(X, Z, S) \doteq \mathsf{dh}(X, Z) \in S$.

An efficient reduction from SCDH to CDH. We have formalized the proof following the approach outlined by Cash et al. in [19]. Note that our proof critically relies on the possibility to relate the probability that an adversary who is called twice wins both times to the probability for a single win. Support for this type of reasoning is a recent extension to EasyCrypt. The proof can be found in Appendix A.

Theorem 2 *Let $\mathcal{A}$ be an SCDH adversary that returns a set of size at most m. Then there exists a CDH adversary $\mathcal{B}$ such that*

$$\Pr\left[\mathsf{SCDH}(\mathcal{A}) = 1\right] \leq \sqrt{\Pr\left[\mathsf{CDH}(\mathcal{B}) = 1\right] + m^2/q}.$$

Furthermore, the adversary $\mathcal{B}$ runs in time $\mathcal{O}(T_{\mathcal{A}} + m^2\, t_{\mathbb{G}})$.

5 Case Studies

We first present the application of our generic proof to the Naxos and Naxos+ protocols. Afterwards, we present our proofs for the Nets protocol.

5.1 Proofs for Naxos and Naxos+

We first prove that Naxos is secure in our $\mathsf{eCK}_{\mathsf{nkr}}$ model with honestly generated keys under the CDH assumption. In the proof, we discuss why it does not generalize to the $\mathsf{eCK}_{\mathsf{kr}}$ model with adversarial key registration. Afterwards, we describe two ways to obtain a proof in the $\mathsf{eCK}_{\mathsf{kr}}$ model from the instantiation of our generic proof. First, the proof can be performed with respect to the Gap-CDH assumption. Second, the protocol can be extended with an additional group element in the key yielding the Naxos+ protocol [39] which was proved secure under the CDH assumption in a model similar to $\mathsf{eCK}_{\mathsf{kr}}$.

$\mathsf{eCK}_{\mathsf{nkr}}$-security of Naxos under the CDH assumption. The following theorem states that Naxos is secure in our model without adversarial key registration if the CDH problem is hard.

Theorem 3 *Let $\mathcal{A}$ be a $(q_{se}, q_{ag}, q_{\boldsymbol{H}})$ $\mathsf{eCK}_{\mathsf{nkr}}$ adversary. Then there exist adversaries $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ such that*

$$\begin{aligned} 2\Pr\left[\mathsf{eCK}_{\mathsf{nkr},\mathsf{Naxos}}(\mathcal{A})=1\right]-1 \leq \quad & \delta_1\ \left(\Pr\left[\mathsf{DLOG}(\mathcal{C}_1)=1\right] + {}^{q_H\, q_{se}}/_{p}\right) \\ & + \delta_2\ \left(\sqrt{\Pr\left[\mathsf{CDH}(\mathcal{C}_2)=1\right] + {}^{4\,q_H^2}/_{p}} + {}^{q_H\, q_{se}}/_{p}\right) \\ & + \delta_3\ \left(\sqrt{\Pr\left[\mathsf{CDH}(\mathcal{C}_3)=1\right] + {}^{4\,q_H^2}/_{p}}\right) + \epsilon_{\mathcal{T}^{\mathsf{nt},\mathsf{hsk}}} \end{aligned}$$

where δ_1, δ_2, δ_3, and $\epsilon_{\mathcal{T}^{\mathsf{nt},\mathsf{hsk}}}$ are defined as in Theorem 1. Furthermore, $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ run in time at most $\mathcal{O}(n\, t_{\mathbb{G}} + t_{\mathcal{A}})$ where $n = \max\{q_h, q_{ag}, q_H\, q_{se}, q_H^2\}$.

Proof. The definition of Naxos^{core} is given in Figure 3. It is easy to check that Naxos^{core} satisfies **P1**–**P3** and $\mathsf{Naxos} = \mathcal{T}^{\mathsf{nt},\mathsf{hsk}}(\mathsf{Naxos}^{core})$. We can therefore apply Theorem 1 to reduce $\mathsf{eCK}_{\mathsf{nkr}}$-security of Naxos to $\mathsf{CSK}^{\mathsf{nt}}_{nkr}$-security of Naxos^{core}. This step accounts for the loss of $\epsilon_{\mathcal{T}^{\mathsf{nt},\mathsf{hsk}}}$ and yields adversaries $\mathcal{B}_1$–$\mathcal{B}_3$ that return sets of size at most $2\,q_H$ and perform at most $q_H\, q_{se}$ queries to eqS. In the next step, we will define $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ and prove that the inequalities

$$\begin{aligned} \Pr\left[G^{\mathsf{hsk},\mathsf{nt}}_{1,\mathsf{nkr}}(\mathcal{B}_1)=1\right] &\leq \Pr\left[\mathsf{DLOG}(\mathcal{C}_1)=1\right] + {}^{q_H\, q_{se}}/_{p} \\ \Pr\left[G^{\mathsf{hsk},\mathsf{nt}}_{2,\mathsf{nkr}}(\mathcal{B}_2)=1\right] &\leq \sqrt{\Pr\left[\mathsf{CDH}(\mathcal{C}_2)=1\right] + {}^{4\,q_H^2}/_{p}} + {}^{q_H\, q_{se}}/_{p} \\ \Pr\left[G^{\mathsf{hsk},\mathsf{nt}}_{3,\mathsf{nkr}}(\mathcal{B}_3)=1\right] &\leq \sqrt{\Pr\left[\mathsf{CDH}(\mathcal{C}_3)=1\right] + {}^{4\,q_H^2}/_{p}} \end{aligned}$$

hold where $G^{\mathsf{hsk},\mathsf{nt}}_{i,\mathsf{nkr}}$ denotes the corresponding $\mathsf{CSK}^{\mathsf{nt}}_{\mathsf{nkr}}$ game instantiated with Naxos^{core}.

Game $G^{\mathsf{hsk},\mathsf{nt}}_{1,\mathsf{nkr}}$. Instantiating with Naxos^{core} yields:

$$\begin{array}{l} a \xleftarrow{\$} \mathbb{F}_p; \quad A \leftarrow g^a \\ \boldsymbol{z} \xleftarrow{\$} \mathbb{F}_p^{q_{se}};\ \boldsymbol{Z} \leftarrow g^{\boldsymbol{z}} \\ \boldsymbol{c} \xleftarrow{\$} \mathbb{F}_p^{q_{ag}-1} \\ a' \leftarrow \mathcal{B}_1(A, \boldsymbol{Z}, \boldsymbol{c}) \\ \mathsf{return}\ (a = a') \end{array}$$

$$\begin{array}{l} eqS(i, Y, j, \hat{A}, \hat{C}, k) = \\ \quad \mathsf{return}\ k = (\mathsf{dh}(A, Y) \,\|\, Z_i^{c_j} \,\|\, \mathsf{dh}(Z_i, Y) \,\|\, \hat{A} \,\|\, \hat{C}) \end{array}$$

Since we perform a reduction to $\mathsf{DLOG}_{\mathsf{2DDH}}$ in the first step, we have already rewritten eqS such that it does not use a and $\boldsymbol{z}$. Before continuing, we rename $\mathsf{DLOG}_{\mathsf{2DDH}}$ such that X becomes A, $\boldsymbol{Y}$ becomes $\boldsymbol{Z}$, and Z becomes R. Our $\mathsf{DLOG}_{\mathsf{2DDH}}$ adversary $\mathcal{C}_1'$ then receives the DLOG-challenge A, the twins $\boldsymbol{Z}$ of A,

and the value R which is unused in DLOG. $\mathcal{C}_1'$ samples $\boldsymbol{c}$, calls $\mathcal{B}_1$ with $(A, \boldsymbol{Z}, \boldsymbol{c})$, and returns $\mathcal{B}_1$'s return value, which is equal to $\mathsf{dlog}(A)$ whenever $\mathcal{B}_1$ wins. $\mathcal{C}_1'$ uses the following implementation to simulate eqS:

$$
\begin{array}{l}
eqS(i, Y, j, \hat{A}, \hat{C}, k) = \\
\quad (U_1 \,\|\, U_2 \,\|\, U_3 \,\|\, \hat{D} \,\|\, \hat{E}) \leftarrow k \\
\quad d \leftarrow \mathsf{2DDH}(i, U_1, U_3) \\
\quad \mathsf{return}\ (d \wedge U_2 = Z_i^{c_j} \wedge \hat{D} = \hat{A} \wedge \hat{E} = \hat{C})
\end{array}
$$

Since the original eqS returns 1 if and only if $U_1 = \mathsf{dh}(A, Y)$ and $U_3 = \mathsf{dh}(Z_i, Y)$ (which corresponds to the $\mathsf{2DDH}$ result) and the remaining equalities hold, the simulation is perfect. We can now apply Lemma 2 to obtain a reduction to DLOG for an adversary $\mathcal{C}_1$.

While this reasoning step is valid in the $\mathsf{eCK_{kr}}$ model, it does not work in the $\mathsf{eCK_{kr}}$ model since the adversary can register arbitrary static public keys. Hence, the eqS oracle takes $C \in \mathsf{Pk}$ instead of an index j into $\boldsymbol{c}$. In this case, we cannot check if $U_2 = \mathsf{dh}(Z_i, C)$ by performing the test $U_2 = Z_i^{c_j}$ in the implementation of eqS for the simulator.

Game $G_{2,\mathsf{nkr}}^{\mathsf{hsk,nt}}$. Instantiating with Naxos^{core} yields:

$$
\begin{array}{ll}
x \xleftarrow{\$} \mathbb{F}_p; & X \leftarrow g^x \\
b \xleftarrow{\$} \mathbb{F}_p; & B \leftarrow g^b \\
\boldsymbol{z} \xleftarrow{\$} \mathbb{F}_p^{q_{se}-1}; & \boldsymbol{Z} \leftarrow g^{\boldsymbol{z}} \\
\boldsymbol{c} \xleftarrow{\$} \mathbb{F}_p^{q_{ag}-1} & \\
(i, Y, \hat{A}, \hat{B}, S) \leftarrow \mathcal{B}_2(X, B, \boldsymbol{c}, \boldsymbol{Z}) & \\
\mathsf{return}\ (Y^{c_i} \,\|\, \mathsf{dh}(B, X) \,\|\, \mathsf{dh}(X, Y) \,\|\, \hat{A} \,\|\, \hat{B}) \in S &
\end{array}
$$

$$
\begin{array}{l}
eqS(j, W, u, \hat{B}, \hat{C}, k) = \\
\quad \mathsf{return}\ k = (\mathsf{dh}(B, W) \,\|\, Z_j^{c_u} \,\|\, \mathsf{dh}(Z_j, W) \,\|\, \hat{B} \,\|\, \hat{C})
\end{array}
$$

For this game, we perform the reduction in three steps. The first reduction is to $\mathsf{SCDH_{2DDH}}$ for which we define the adversary $\mathcal{C}_2'$. Then we use Lemma 2 to get rid of the $\mathsf{2DDH}$ oracle and finally Theorem 2 to transform the SCDH adversary into a CDH adversary which yields the adversary $\mathcal{B}_2$. Before continuing, we rename $\mathsf{SCDH_{2DDH}}$ such that X becomes B, $\boldsymbol{Y}$ becomes $\boldsymbol{Z}$, and Y becomes X. The CDH challenge is therefore B, X and $\boldsymbol{Z}$ is the vector of twins of B for which the $\mathsf{2DDH}$ oracle can be used.

We define the $\mathsf{SCDH_{2DDH}}$ adversary $\mathcal{C}_2'$ as follows. $\mathcal{C}_2'$ gets $B, \boldsymbol{Z}, X$ as input, samples $\boldsymbol{c}$, calls $\mathcal{B}_2$ with these values, and gets $(i, Y, \hat{A}, \hat{B}, S)$. To transform S into a set that contains $\mathsf{dh}(B, X)$ whenever $\mathcal{B}_2$ wins, $\mathcal{C}_2'$ applies the function $(U_1 \,\|\, U_2 \,\|\, U_3 \,\|\, \hat{A} \,\|\, \hat{B}) \mapsto U_2$. To (perfectly) simulate the orginal eqS, $\mathcal{C}_2'$ uses the implementation

$$
\begin{array}{l}
eqS(j, W, u, \hat{B}, \hat{C}, k) = \\
\quad (U_1 \,\|\, U_2 \,\|\, U_3 \,\|\, \hat{D} \,\|\, \hat{E}) \leftarrow k \\
\quad d \leftarrow \mathsf{2DDH}(i, U_1, U_3) \\
\quad \mathsf{return}\ (d \wedge U_2 = Z_j^{c_u} \wedge \hat{D} = \hat{B} \wedge \hat{E} = \hat{C}).
\end{array}
$$

Since the adversary can register arbitrary static public keys in the $\mathsf{eCK}_{\mathsf{kr}}$ model, the *eqS* oracle in the kr version of this game takes $C \in \mathsf{Pk}$ instead of an index u into $\boldsymbol{c}$. In this case, we cannot check if $U_2 = \mathsf{dh}(Z_j, C)$ by performing the test $U_2 = Z_j^{c_u}$ in the implementation of *eqS* for the simulator.

Game $G^{\mathsf{hsk},\mathsf{nt}}_{3,\mathsf{nkr}}$. Instantiating with Naxos^{core} yields:

$$
\begin{array}{l}
x \xleftarrow{\$} \mathbb{F}_p;\ X \leftarrow g^x \\
y \xleftarrow{\$} \mathbb{F}_p;\ Y \leftarrow g^y \\
\boldsymbol{c} \xleftarrow{\$} \mathsf{Sk}^{q_{ag}} \\
(i, j, \hat{A}, \hat{B}, S) \leftarrow \mathcal{B}_3(X, Y, \boldsymbol{c}) \\
\mathsf{return}\ (Y^{c_i} \,\|\, X^{c_j} \,\|\, \mathsf{dh}(X, Y) \,\|\, \hat{A} \,\|\, \hat{B}) \in S
\end{array}
$$

We can directly perform a reduction to SCDH and then use Theorem 2 to obtain a reduction to CDH. For the reduction to SCDH, we use the function $(U_1 \,\|\, U_2 \,\|\, U_3 \,\|\, \hat{D} \,\|\, \hat{E}) \mapsto U_3$ to transform S into a set that contains $\mathsf{dh}(X, Y)$. This case directly generalizes to $\mathsf{eCK}_{\mathsf{kr}}$ since the third game is identical in this case. □

$\mathsf{eCK}_{\mathsf{kr}}$-security of Naxos and Naxos$^+$. In the previous proof, we have pointed out where the proof breaks down in the $\mathsf{eCK}_{\mathsf{kr}}$ model. We will now describe how to adapt the proof to (1) prove $\mathsf{eCK}_{\mathsf{kr}}$-security of Naxos under the Gap-CDH assumption and (2) prove $\mathsf{eCK}_{\mathsf{kr}}$-security of Naxos^+ under the CDH assumption.

For the proof with respect to Gap-CDH, we can deal with all the problematic cases by calling the DDH oracle with the right input, e.g., with $\mathsf{DDH}(Z_i, C, U_2)$ for the first game. Note that there is no need for the twinning technique at all in this case and our generic proof can be instantiated in a very similar way to the original Naxos proof.

The Naxos^{+core} protocol can be obtained from the Naxos^{core} protocol by adding the additional group element $\mathsf{dh}(A, B)$ to the key. Concretely, we define

$$
\begin{array}{l}
\mathsf{Key} = \mathbb{G}^4 \times \mathsf{ID}^2, \\
\mathit{KeyI}(x, a, Y, B, \hat{A}, \hat{B}) = Y^a \,\|\, B^x \,\|\, Y^x \,\|\, B^a \,\|\, \hat{A} \,\|\, \hat{B}, \text{ and} \\
\mathit{KeyR}(y, b, X, A, \hat{B}, \hat{A}) = A^y \,\|\, X^b \,\|\, X^y \,\|\, A^b \,\|\, \hat{A} \,\|\, \hat{B}.
\end{array}
$$

The additional group element is only required to simulate the *eqS* oracle. Everything else, in particular the case **Game** $G^{\mathsf{hsk},\mathsf{nt}}_3$, can be trivially adapted.

Game $G^{\mathsf{hsk},\mathsf{nt}}_{1,\mathsf{kr}}$. For Naxos^+, we must simulate the following *eqS* oracle (we underline the differences to the Naxos version):

$$
\begin{array}{l}
\mathit{eqS}(i, Y, \underline{C}, \hat{A}, \hat{C}, k) = \\
\quad \mathsf{return}\ k = (\mathsf{dh}(A, Y) \,\|\, \underline{\mathsf{dh}(Z_i, C)} \,\|\, \mathsf{dh}(Z_i, Y) \,\|\, \underline{\mathsf{dh}(A, C)} \,\|\, \hat{A} \,\|\, \hat{C})
\end{array}
$$

By using the 2DDH oracle for the group elements 1&3 and 2&4, we obtain the following implementation of *eqS*.

$$\begin{array}{l} eqS(i, Y, j, \hat{A}, \hat{C}, k) = \\ \quad (U_1 \,\|\, U_2 \,\|\, U_3 \,\|\, U_4 \,\|\, \hat{D} \,\|\, \hat{E}) \leftarrow k \\ \quad d_1 \leftarrow \mathsf{2DDH}(i, U_1, U_3) \\ \quad d_2 \leftarrow \mathsf{2DDH}(i, U_2, U_4) \\ \quad \mathsf{return}\ (d_1 \wedge d_2 \wedge \hat{D} = \hat{A} \wedge \hat{E} = \hat{C}) \end{array}$$

The simulation is perfect because the 2DDH calls returns 1 if and only if $U_1 = \mathsf{dh}(A, Y) \wedge U_3 = \mathsf{dh}(Z_i, Y)$ and $U_4 = \mathsf{dh}(A, C) \wedge U_2 = \mathsf{dh}(Z_i, C)$.

Game $G^{\mathsf{hsk,nt}}_{2,\mathsf{kr}}$. For Naxos+, we must simulate the following *eqS* oracle

$$\begin{array}{l} eqS(j, W, \underline{C}, \hat{B}, \hat{C}, k) = \\ \quad \mathsf{return}\ k = (\mathsf{dh}(B, W) \,\|\, \underline{\mathsf{dh}(Z_j, C)} \,\|\, \mathsf{dh}(Z_j, W) \,\|\, \underline{\mathsf{dh}(B, C)} \,\|\, \hat{B} \,\|\, \hat{C}) \end{array}$$

By using the 2DDH oracle first for the group elements 1 and 3 and then using the oracle in a second call for the group elements 2 and 4, we obtain the following implementation of *eqS*.

$$\begin{array}{l} eqS(i, Y, j, \hat{A}, \hat{C}, k) = \\ \quad (U_1 \,\|\, U_2 \,\|\, U_3 \,\|\, U_4 \,\|\, \hat{D} \,\|\, \hat{E}) \leftarrow k \\ \quad d_1 \leftarrow \mathsf{2DDH}(i, U_1, U_3) \\ \quad d_2 \leftarrow \mathsf{2DDH}(i, U_4, U_2) \\ \quad \mathsf{return}\ (d_1 \wedge d_2 \wedge \hat{D} = \hat{A} \wedge \hat{E} = \hat{C}) \end{array}$$

The simulation is perfect because the first 2DDH calls returns 1 iff $U_1 = \mathsf{dh}(B, W) \wedge U_3 = \mathsf{dh}(Z_j, W)$ and the second call returns 1 iff $U_4 = \mathsf{dh}(B, C) \wedge U_2 = \mathsf{dh}(Z_j, C)$.

5.2 Proofs for Nets

The proofs for Nets are structured very similarly to the corresponding Naxos proofs and yield similar bounds. We therefore summarize the required changes in this section and refer to our EasyCrypt formalization for details.

The proof that Nets is secure in our $\mathsf{eCK}_{\mathsf{nkr}}$ model with honestly generated keys under the CDH assumption is follows the corresponding proof for Naxos. The only significant difference is how the 2DDH oracle is used to simulate the *eqS* oracles in the first and second games. Whereas the Naxos protocol uses the concatenation of three group elements in the key, Nets uses the concatenation of two group elements $U_1 \,\|\, U_2$ where $U_1 = \mathsf{dh}(A, B)\,\mathsf{dh}(A, Y)\,\mathsf{dh}(X, B)\,\mathsf{dh}(X, Y)$ and $U_2 = cdh(X, Y)$. Computing the right queries to 2DDH for simulating *eqS* requires divisions. Concretely, the first game uses $\mathsf{2DDH}(U_1/A^{c_j}\ Z_i^{c_j}\ U_2, U_2)$ and the second game uses $\mathsf{2DDH}(U_1/B^{c_j}\ Z_i^{c_j}\ U_2, U_2)$.

To prove $\mathsf{eCK}_{\mathsf{kr}}$-security under the Gap-CDH assumption, it is again possible to closely follow the original proof and use the DDH oracle to simulate *eqS*.

G_1^{hsk} (a, b secret, poss. no matching):	G_2^{hsk} (x, b secret, poss. no matching):
$a, b \xleftarrow{\$} \mathsf{Sk};\ A \leftarrow Pk(a);\ B \leftarrow Pk(b)$ $\boldsymbol{z} \xleftarrow{\$} \mathsf{Esk}^{q_{se}}$ $(i, Y, \hat{A}, \hat{B}, S) \leftarrow \mathcal{A}(\boldsymbol{z}, A, B)$ $k \leftarrow KeyI(z_i, a, Y, B, \hat{A}, \hat{B})$ $\mathsf{return}\ (k \in S \wedge k \neq \perp)$ $eqS(i, C, W, D, \hat{C}, \hat{D}, k) =$ $\quad \mathsf{if}\ C \notin \{A, B\}\ \mathsf{then\ return}\ \perp$ $\quad \mathsf{if}\ C = A\ \mathsf{then}\ c \leftarrow a\ \mathsf{else}\ c \leftarrow b$ $\quad \mathsf{return}\ KeyI(z_i, c, W, D, \hat{C}, \hat{D}) = k$	$x \xleftarrow{\$} \mathsf{Esk};\ X \leftarrow Epk(X)$ $b \xleftarrow{\$} \mathsf{Sk};\quad B \leftarrow Pk(b)$ $\boldsymbol{z} \xleftarrow{\$} \mathsf{Esk}^{q_{se}-1}$ $\boldsymbol{c} \xleftarrow{\$} \mathsf{Sk}^{q_{ag}-1}$ $(i, Y, \hat{A}, \hat{B}, S) \leftarrow \mathcal{A}(\boldsymbol{c}, \boldsymbol{z}, X, B)$ $k \leftarrow KeyI(x, c_i, Y, B, \hat{A}, \hat{B})$ $\mathsf{return}\ (k \in S \wedge k \neq \perp)$ $eqS(i, W, D, \hat{B}, \hat{D}, k) =$ $\quad \mathsf{return}\ KeyI(z_i, b, W, D, \hat{B}, \hat{D}) = k$
G_3^{hsk} (a, y secret):	G_4^{hsk} (x, y secret):
$a \xleftarrow{\$} \mathsf{Sk};\ A \leftarrow Pk(a)$ $y \xleftarrow{\$} \mathsf{Esk};\ Y \leftarrow Epk(y)$ $\boldsymbol{z} \xleftarrow{\$} \mathsf{Esk}^{q_{se}-1};\ \boldsymbol{c} \xleftarrow{\$} \mathsf{Sk}^{q_{ag}-1}$ $(i, j, \hat{A}, \hat{B}, S) \leftarrow \mathcal{A}(\boldsymbol{c}, \boldsymbol{z}, A, Y)$ $k \leftarrow KeyI(z_i, a, Y, Pk(c_j), \hat{A}, \hat{B})$ $\mathsf{return}\ (k \in S \wedge k \neq \perp)$ $eqS(i, W, D, \hat{A}, \hat{D}, k) =$ $\quad \mathsf{return}\ KeyI(z_i, a, W, D, \hat{A}, \hat{D}) = k$	$x \xleftarrow{\$} \mathsf{Esk};\ X \leftarrow Epk(x)$ $y \xleftarrow{\$} \mathsf{Esk};\ Y \leftarrow Epk(y)$ $\boldsymbol{c} \xleftarrow{\$} \mathsf{Sk}^{q_{ag}}$ $(i, j, \hat{A}, \hat{B}, S) \leftarrow \mathcal{A}(\boldsymbol{c}, X, Y)$ $k \leftarrow KeyI(x, c_i, Y, Pk(c_j), \hat{A}, \hat{B})$ $\mathsf{return}\ (k \in S \wedge k \neq \perp)$

Fig. 9. Games defining $\mathsf{CSK}_{\mathsf{kr}}$.

6 Protocols Without Naxos Trick

In this section, we describe our generic proof for protocols that do no utilize the Naxos trick and its application to a version of HMQV. The results of this section have not been formalized in EasyCrypt and we leave this open for future work.

6.1 Model and Generic Proof

We prove a reduction from the $\mathsf{eCK}_{\mathsf{kr}}$ model to the $\mathsf{CSK}_{\mathsf{kr}}$ model defined by the games given in Figure 9.

Theorem 4 *Let Π be a protocol that satisfies* **P1**–**P3**. *For all efficient adversaries that win the $\mathsf{eCK}_{\mathsf{kr},\mathcal{T}^{\mathsf{hsk}}(\Pi)}$ game with non-negligible probability, there exists an efficient adversary that wins one of the $\mathsf{CSK}_{\mathsf{kr},\Pi}$ games with non-negligible probability .*

The proof is analogous to the proof of Theorem 1 and appears in the full version of the paper [4]. The proof performs a different case distinction with respect to the reveal queries performed by the adversary than the proof of Theorem 1.

6.2 $\mathsf{eCK}_{\mathsf{kr}}$-security of mHMQV under the Gap-CDH assumption

We first define our (modified version) mHMQV^{core} as follows. We use $Pk(a) = g^a$ and $Epk(x, a) = g^x$ for ephemeral and static key computation. Using the hash function $\bar{h} : \mathbb{G} \to \mathbb{F}_p$, we define the session keys:

$$KeyI(x, a, Y, B, \hat{A}, \hat{B}) = (Y B^{\bar{h}(Y)})^{x + a\,\bar{h}(X)} \,\|\, \hat{A} \,\|\, \hat{B} \,\|\, X \,\|\, Y$$
$$KeyR(y, b, X, A, \hat{B}, \hat{A}) = (X A^{\bar{h}(X)})^{y + b\,\bar{h}(Y)} \,\|\, \hat{A} \,\|\, \hat{B} \,\|\, X \,\|\, Y$$

We instantiate the types using $\mathbb{G}$ for group elements and $\mathbb{F}_p$ for exponents. We then define mHMQV as $\mathcal{T}^{\mathsf{hsk}}(\mathsf{mHMQV}^{core})$. A similar version of HMQV has been proposed in the original paper [31, Remark 9.1] for compatibility between the variants with one, two, and three passes. We slightly deviate from the original definition by removing the identities from $\bar{h}$'s input (like in MQV) and including $\hat{A}$, $\hat{B}$, X and Y as input to the key derivation hash. Including additional session data in the hash is considered a prudent engineering principle [16] because it ensures agreement on this data. Second, it allows us to apply our generic proof directly since the resulting protocol satisfies **P3**. To make the protocol symmetric, it would be possible to sort the tuples $\hat{A}, X$ and $\hat{B}, Y$ to determine the order of these elements. We prove the following theorem for mHMQV.

Theorem 5 *If there is an efficient adversary that wins the* $\mathsf{eCK}'_{\mathsf{mHMQV}}$ *game with non-negligible probability, then there is an efficient adversary that wins the Gap-CDH game with non-negligible probability.*

Proof (Sketch). Since mHMQV^{core} satisfies **P1**–**P3**, we can apply Theorem 4 and prove $\mathsf{CSK}_{\mathsf{kr}}$-security of mHMQV^{core}. As in the Nets proof, we ignore the public part $\hat{A} \,\|\, \hat{B} \,\|\, X \,\|\, Y$ in our discussion of winning conditions and *eqS*. Before discussing the individual games, we note that under the Gap-CDH assumption which provides a DDH-oracle, it is possible to simulate the *eqS* oracle in all of the games since at least the secret key z_i is always known. To simulate *eqS* queries, e.g., in G_1^{hsk}, it suffices to compute

$$W^{z_i}\, \mathsf{dh}(C, W)^{\bar{h}(g^{z_i})}\, D^{z_i\,\bar{h}(W)}\, \mathsf{dh}(C, D)^{\bar{h}(g^{z_i})\,\bar{h}(W)} = k$$

for z_i in $\boldsymbol{z}$, $C \in \{A, B\}$, and W, D, k arbitrary. To achieve this, the DDH oracle can be used to check

$$\mathsf{dh}(C, W^{\bar{h}(g^{z_i})} D^{\bar{h}(g^{z_i})\,\bar{h}(W)}) = \frac{k}{W^{z_i}\, E^{z_i\,\bar{h}(W)}}.$$

For game G_1^{hsk}, we perform a reduction to Gap-CDH using the Forking Lemma. We know there exists an adversary $\mathcal{A}$ such that for the CDH challenge A, B and uniformly sampled $\boldsymbol{z}$, the call $\mathcal{A}(\boldsymbol{z}, A, B)$ returns i, Y, and a set S that contains

$$Y^{z_i}\, \mathsf{dh}(A, Y)^{\bar{h}(Z_i)}\, B^{z_i\,\bar{h}(Y)}\, \mathsf{dh}(A, B)^{\bar{h}(Z_i)\,\bar{h}(Y)}$$

with non-negligible probability. To apply the Forking Lemma from [9], we use $\mathcal{A}$ to define a randomized algorithm $\mathcal{B}$ that returns $v \in [q_{\bar{h}}]$, $\mathsf{dh}(A, Y)\, \mathsf{dh}(A, B)^{\bar{h}(Y)}$,

and Y such that Y is the v-th query to $\bar{h}$ with non-negligible probability. First, $\mathcal{B}$ guesses v, then it calls $\mathcal{A}$ with the CDH challenge A, B and uniformly sampled $\boldsymbol{z}$. $\mathcal{B}$ then computes the result from $\mathcal{A}$'s return values i, Y, S as follows. If Y is not the v-th query, $\mathcal{B}$ returns $\perp$. Otherwise, $\mathcal{B}$ divides all elements of S by $Y^{z_i}\, B^{z_i\, \bar{h}(Y)}$, exponentiates the result with $1/\bar{h}(g^{z_i})$, and uses the DDH-oracle to search for U with $\mathsf{ddh}(A, Y\, B^{\bar{h}(Y)}, U)$. If there is no such value, $\mathcal{B}$ returns $\perp$. Otherwise, $\mathcal{B}$ returns $v, Y, \mathsf{dh}(A, Y)\, \mathsf{dh}(A, B)^{\bar{h}(Y)}$. The Forking Lemma yields a randomized algorithm $\mathcal{C}$ from $\mathcal{B}$ that returns

$$Y, \mathsf{dh}(A, Y)\, \mathsf{dh}(A, B)^{e}, \mathsf{dh}(A, Y)\, \mathsf{dh}(A, B)^{e'}$$

with $e \neq e'$ with non-negligible probability. Intuitively, the algorithm first calls $\mathcal{B}$ to obtain $v, Y, \mathsf{dh}(A, Y)\, \mathsf{dh}(A, B)^{e}$ for $e = \bar{h}(Y)$. Then, it calls $\mathcal{B}$ again using the same randomness, but resampling the values returned by $\bar{h}$ for all query-indices greater or equal than v, i.e., $e' = \bar{h}(Y)$ is the first value that differs. We can then compute

$$\mathsf{dh}(A, B) = \left(\frac{\mathsf{dh}(A, Y)\, \mathsf{dh}(A, B)^{e}}{\mathsf{dh}(A, Y)\, \mathsf{dh}(A, B)^{e'}} \right)^{\frac{1}{e - e'}} .$$

For game G_2^{hsk}, we also reduce to Gap-CDH. We know there exists an adversary $\mathcal{A}$ such that for the CDH challenge X, B and uniformly sampled $\boldsymbol{c}$ and $\boldsymbol{z}$, the call $\mathcal{A}(\boldsymbol{c}, \boldsymbol{z}, X, B)$ returns i, Y, and a set S that contains

$$\mathsf{dh}(X, Y)\, Y^{c_i \bar{h}(X)}\, \mathsf{dh}(X, B)^{\bar{h}(Y)}\, B^{c_i\, \bar{h}(X)\, \bar{h}(Y)}$$

with non-negligible probability. Using a similar approach as before, we can obtain an algorithm that returns the group element $\mathsf{dh}(X, Y)\, \mathsf{dh}(X, B)^{e}$ and the group element $\mathsf{dh}(X, Y)\, \mathsf{dh}(X, B)^{e'}$ for $e \neq e'$ with non-negligible probability. We can then compute $\mathsf{dh}(X, B)$ like in the previous case.

For G_3^{hsk}, the reduction to Gap-CDH is simpler than the previous two cases since we know two secret keys instead of only one. We can call $\mathcal{A}$ with randomly sampled $\boldsymbol{c}, \boldsymbol{z}$, and a CDH challenge A, Y. Since $\mathcal{A}$ returns i, j, and a set S that contains

$$Y^{z_i}\, \mathsf{dh}(A, Y)^{\bar{h}(g^{z_i})}\, g^{z_i\, c_j\, \bar{h}(Y)}\, A^{c_j\, \bar{h}(g^{z_i})\, \bar{h}(Y)}$$

with non-negligible probability, we can then use the DDH oracle to find $\mathsf{dh}(A, Y)$. For G_4^{hsk}, we can perform a similar reduction to Gap-CDH for the CDH challenge X, Y. □

Acknowledgement

We thank the anonymous reviewers for their valuable comments and suggestions. We also thank François Dupressoir, Benjamin Grégoire, César Kunz, and Pierre-Yves Strub for their help and the development of $\mathsf{EasyCrypt}$ features required to build the proof. This work is supported in part by ONR grant N00014-12-1-0914, Madrid regional project S2009TIC-1465 PROMETIDOS, and Spanish projects TIN2009-14599 DESAFIOS 10, and TIN2012-39391-C04-01 Strongsoft. The research of Schmidt has received funds from the European Commission's Seventh

Framework Programme Marie Curie Cofund Action AMAROUT II (grant no. 291803).

References

1. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
2. J. B. Almeida, M. Barbosa, G. Barthe, and F. Dupressoir. Certified computer-aided cryptography: efficient provably secure machine code from high-level implementations. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 1217–1230. ACM Press, Nov. 2013.
3. G. Barthe, J. M. Crespo, B. Grégoire, C. Kunz, Y. Lakhnech, B. Schmidt, and S. Z. Béguelin. Fully automated analysis of padding-based encryption in the computational model. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 1247–1260. ACM Press, Nov. 2013.
4. G. Barthe, J. M. Crespo, Y. Lakhnech, and B. Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. Cryptology ePrint Archive 2015, 2015. `http://eprint.iacr.org/`.
5. G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub. Easycrypt: A tutorial. In A. Aldini, J. Lopez, and F. Martinelli, editors, *Foundations of Security Analysis and Design VII*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer International Publishing, 2014.
6. G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin. Computer-aided security proofs for the working cryptographer. In P. Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, Aug. 2011.
7. D. A. Basin and C. J. F. Cremers. Modeling and analyzing security in the presence of compromising adversaries. In D. Gritzalis, B. Preneel, and M. Theoharidou, editors, *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 340–356. Springer, 2010.
8. M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *30th Annual ACM Symposium on Theory of Computing*, pages 419–428. ACM Press, May 1998.
9. M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In A. Juels, R. N. Wright, and S. Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 390–399. ACM Press, Oct. / Nov. 2006.
10. M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, Aug. 1993.
11. N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004*, pages 14–25, New York, 2004. ACM.
12. K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P. Strub. Implementing tls with verified cryptographic security. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 445–459. IEEE, 2013.

13. K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, and S. Z. Béguelin. Proving the TLS handshake secure (as it is). In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 235–255, 2014.
14. B. Blanchet. A computationally sound mechanized prover for security protocols. In *2006 IEEE Symposium on Security and Privacy*, pages 140–154. IEEE Computer Society Press, May 2006.
15. B. Blanchet. Security protocol verification: Symbolic and computational models. In *1st International Conference on Principles of Security and Trust, POST 2012*, volume 7215 of *Lecture Notes in Computer Science*, pages 3–29, Heidelberg, 2012. Springer.
16. C. Boyd, C. Cremers, M. Feltz, K. G. Paterson, B. Poettering, and D. Stebila. ASICS: Authenticated key exchange security incorporating certification systems. In J. Crampton, S. Jajodia, and K. Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 381–399. Springer, Sept. 2003.
17. C. Brzuska, N. P. Smart, B. Warinschi, and G. J. Watson. An analysis of the EMV channel establishment protocol. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 373–386. ACM Press, Nov. 2013.
18. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, May 2001.
19. D. Cash, E. Kiltz, and V. Shoup. The twin Diffie-Hellman problem and applications. In N. P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 127–145. Springer, Apr. 2008.
20. K.-K. R. Choo, C. Boyd, and Y. Hitchcock. Examining indistinguishability-based proof models for key establishment protocols. In B. K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 585–604. Springer, Dec. 2005.
21. V. Cortier, S. Kremer, and B. Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *Journal of Automated Reasoning*, 46(3-4):225–259, 2011.
22. C. J. Cremers. Formally and practically relating the ck, ck-hmqv, and eck security models for authenticated key exchange. Cryptology ePrint Archive, Report 2009/253, 2009. `http://eprint.iacr.org/`.
23. C. J. F. Cremers and M. Feltz. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In S. Foresti, M. Yung, and F. Martinelli, editors, *ESORICS 2012: 17th European Symposium on Research in Computer Security*, volume 7459 of *Lecture Notes in Computer Science*, pages 734–751. Springer, Sept. 2002.
24. W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Des. Codes Cryptography*, 2(2):107–125, 1992.
25. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, Aug. 1984.
26. S. Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint Archive, Report 2005/181, 2005. `http://eprint.iacr.org/2005/181`.

27. T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293. Springer, Aug. 2012.
28. B. S. Kaliski Jr. An unknown key-share attack on the MQV key agreement protocol. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):275–288, 2001.
29. M. Kim, A. Fujioka, and B. Ustaoğlu. Strongly secure authenticated key exchange without naxos' approach. In *Advances in Information and Computer Security*, pages 174–191. Springer, 2009.
30. H. Krawczyk. SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols. In *Advances in Cryptology-CRYPTO 2003*, pages 400–425. Springer, 2003.
31. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, Aug. 2005.
32. H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448. Springer, Aug. 2013.
33. C. Kudla and K. G. Paterson. Modular security proofs for key agreement protocols. In B. K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 549–565. Springer, Dec. 2005.
34. C. J. Kudla. *Special Signature Schemes and Key Agreement Protocols*. PhD thesis, University of London, 2006.
35. R. Küsters and T. Truderung. Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In *Computer Security Foundations Symposium (CSF)*, pages 157–171. IEEE, 2009.
36. B. A. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In W. Susilo, J. K. Liu, and Y. Mu, editors, *ProvSec 2007: 1st International Conference on Provable Security*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer, Nov. 2007.
37. L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003.
38. J. Lee and C. S. Park. An efficient authenticated key exchange protocol with a tight security reduction. *IACR Cryptology ePrint Archive*, 2008:345, 2008.
39. J. Lee and J. H. Park. Authenticated key exchange secure under the computational diffie-hellman assumption. Cryptology ePrint Archive, Report 2008/344, 2008. `http://eprint.iacr.org/`.
40. T. Matsumoto and Y. Takashima. On seeking smart public-key-distribution systems. *IEICE TRANSACTIONS (1976-1990)*, 69(2):99–106, 1986.
41. A. Menezes. Another look at HMQV. *Mathematical Cryptology JMC*, 1(1):47–64, 2007.
42. A. Menezes and B. Ustaoglu. On the importance of public-key validation in the MQV and HMQV key agreement protocols. In R. Barua and T. Lange, editors, *Progress in Cryptology - INDOCRYPT 2006: 7th International Conference in Cryptology in India*, volume 4329 of *Lecture Notes in Computer Science*, pages 133–147. Springer, Dec. 2006.

43. T. Okamoto and D. Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In K. Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118. Springer, Feb. 2001.
44. J. Pan and L. Wang. Tmqv: a strongly eck-secure diffie-hellman protocol without gap assumption. In *Provable Security*, pages 380–388. Springer, 2011.
45. W. M. Petullo, X. Zhang, J. A. Solworth, D. J. Bernstein, and T. Lange. MinimaLT: minimal-latency networking through better security. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 425–438. ACM Press, Nov. 2013.
46. B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *Computer Security Foundations Symposium (CSF)*, pages 78–94. IEEE, 2012.
47. V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, May 1997.

A Proofs For Twin DH and (S)CDH

In this appendix, we present the proofs for Lemma 2 and Theorem 2.

Proof (of Lemma 2). We define $\mathcal{B}$ as

$$\begin{array}{l}\mathcal{B}(Z,Y) \doteq \\ \quad \boldsymbol{r} \xleftarrow{\$} \mathbb{F}_p^n;\ \boldsymbol{s} \xleftarrow{\$} \mathbb{F}_p^n;\ Y_1 \leftarrow g^{s_1}/X^{r_1};\ \ldots;\ Y_n \leftarrow g^{s_n}/X^{r_n} \\ \quad \mathsf{return}\ \mathcal{A}^{\mathsf{2DDH}}(X, \boldsymbol{Y}, Z)\end{array}$$

and note that the distribution on $(X, \boldsymbol{Y}, Z)$ is the same as in G_{2DDH}. To simulate the 2DDH oracle, $\mathcal{B}$ uses the test $U^{r_i}V = \hat{Z}^{s_i}$ instead of $\mathsf{ddh}(X, \hat{Z}, U) \wedge \mathsf{ddh}(Y_i, \hat{Z}, V)$. The probability that these tests do not agree is at most $1/p$. Since the adversary can perform q queries to 2DDH, the probability of distinguishing the simulator is at most q/p. □

Proof (Theorem 2). We first prove that

$$\Pr\left[\mathsf{SCDH}(\mathcal{A}) = 1\right] = \sqrt{\Pr\left[\mathsf{CDH}_{\mathsf{2DDH}}(\mathcal{B}) = 1\right]}$$

where $n = 1$ for $\mathsf{CDH}_{\mathsf{2DDH}}$, i.e., there is only one twin. To achieve this, we define:

$$\begin{array}{rl}\mathcal{B}(X,Y,Z) \doteq u \xleftarrow{\$} \mathbb{F}_p^*;\ S_1 \leftarrow \mathcal{A}(X,Z);\ S_2 \leftarrow \mathcal{A}(Y,Z^u) & \\ \quad \mathsf{foreach}\ (Z_1,Z_2) \in S_1 \times S_2 : & \\ \quad\quad \mathsf{if}\ \mathsf{2DDH}(Z,Z_1,Z_2^{1/u})\ \mathsf{then\ return}\ Z_1 & \end{array}$$

Since $\mathcal{B}$ wins whenever $\mathcal{A}$ wins both times, $\mathcal{B}$'s winning probability is equal to the square of $\mathcal{A}$'s winning probability. We then conclude the proof by applying Lemma 2 and observing that the given simulator calls the 2DDH oracle at most m^2 times. □