Flexible Authenticated and Confidential Channel Establishment (fACCE): Analyzing the Noise Protocol Framework*

Benjamin Dowling[1], Paul Rösler[2], and Jörg Schwenk[2]

[1] Applied Cryptography Group, Eidgenössische Technische Hochschule Zürich
benjamin.dowling@inf.ethz.ch
[2] Horst-Görtz Institute for IT Security,
Chair for Network and Data Security, Ruhr University Bochum
{paul.roesler,joerg.schwenk}@rub.de

Abstract. The Noise protocol framework is a suite of channel establishment protocols, of which each individual protocol ensures various security properties of the transmitted messages, but keeps specification, implementation, and configuration relatively simple. Implementations of the Noise protocols are themselves, due to the employed primitives, very performant. Thus, despite its relative youth, Noise is already used by large-scale deployed applications such as WhatsApp and Slack. Though the Noise specification describes and claims the security properties of the protocol patterns very precisely, there has been no computational proof yet. We close this gap.

Noise uses only a limited number of cryptographic primitives which makes it an ideal candidate for reduction-based security proofs. Due to its patterns' characteristics as channel establishment protocols, and the usage of established keys within the handshake, the authenticated and confidential channel establishment (ACCE) model (Jager et al. CRYPTO 2012) seems to perfectly fit for an analysis of Noise. However, the ACCE model strictly divides protocols into two non-overlapping phases: the *pre-accept phase* (i.e., the channel establishment) and *post-accept phase* (i.e., the channel). In contrast, Noise allows the transmission of encrypted messages as soon as any key is established (for instance, before authentication between parties has taken place), and then incrementally increases the channel's security guarantees. By proposing a generalization of the original ACCE model, we capture security properties of such staged channel establishment protocols flexibly – comparably to the multi-stage key exchange model (Fischlin and Günther CCS 2014).

We give security proofs for eight of the 15 basic Noise patterns in the full version (EPRINT 2019/436) and exemplify them by the proof of the `XK` pattern in this article.

Keywords: Noise protocol framework; ACCE; Multi-Stage; Channel Establishment

* The full version of this article is available in the IACR eprint archive as article 2019/436, at https://eprint.iacr.org/2019/436.

1 Introduction

Noise is a protocol framework introduced by Trevor Perrin [32] for establishing confidential channels between two parties in various application scenarios that bases on a Diffie-Hellman group, a secure key derivation function (KDF), a secure hash function, and a secure authenticated encryption with associated data scheme (AEAD). Like TLS 1.2, Noise makes use of the derived keys during channel establishment, which makes an analysis with respect to *key indistinguishability* as in traditional key exchange models infeasible. Furthermore, to allow the transmission of messages as early as possible (to avoid latency costs), protocols like TLS 1.3 and Noise amalgamate handshake and channel (at cost of security guarantees for these early messages). In this work we analyze the security of patterns from the Noise framework and, since previous modeling approaches cannot be used under the aforementioned conditions, introduce the flexible ACCE model to prove fine-grained security guarantees of Noise.

The Noise Framework The Noise protocol framework is a tool box for defining simple and lightweight protocols for homogeneous environments. In this context, homogeneous means that all parties in the environment agree upon the protocol (including mechanisms for long-term key distribution, protocol version, employed cryptographic primitives, ...). In contrast, TLS allows the establishment of a channel in highly federated environments, in which that information has not been agreed upon by the protocol participants. This induces highly complex implementations that contain version and cipher suite negotiation as well as legacy code. Noise can disregard these issues (which in TLS regularly lead to security vulnerabilities, e.g., [1, 31]) but still offers multiple protocol patterns that allow a developer to choose a protocol fulfilling their application's security needs and considering the respective use case (long-term key distribution, latency, ...).

The Noise specification defines 15 core protocol patterns for different use-cases, which may consist of one, two, or three handshake messages (cf. Figure 1) – containing ephemeral and/or long-term Diffie-Hellman shares and (if a key is already established) an AEAD ciphertext – and a channel. Each party can have a long-term DH key pair, and potentially contributes one ephemeral DH key share per protocol execution. The different patterns of Noise can hence be seen as different distributions of the corresponding two to four public DH shares to the handshake messages. The three-message patterns of Noise are novel in the sense that classical three/four-message patterns for AKE protocols typically use only one DH key exchange which is either static (TLS-DH) or ephemeral (signed DH, Station-to-Station protocol, TLS 1.3, TLS-DHE, IPsec IKE, SSH) combined with digital signatures (all of the above) or MACs (IPsec IKE Phase 2 with forward-secrecy). Noise avoids authentication with MACs or digital signatures, and provides entity authentication via long-term DH keys, key derivation, and AEAD ciphertexts.

As a result, Noise is for its scope even more agile than TLS, allowing tailored protocols for multiple use-cases with various security properties. Resulting from

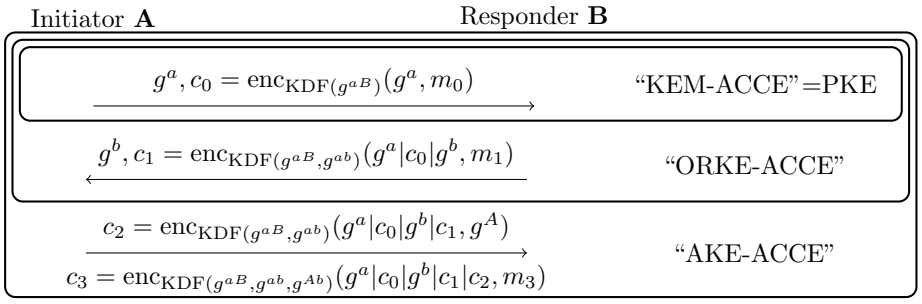

Fig. 1: The flexible structure of the Noise protocol framework, described conceptually with the `XK` pattern (three passes) that is based on the `NK` pattern (two passes), which is based on the `N` pattern (one pass). g^A and g^B denote the long-term public DH shares of parties **A** and **B**, g^a and g^b denote their ephemeral shares, and $\text{enc}_k(ad, m)$ is an AEAD encryption.

its efficiency and flexibility, Noise is used by largely deployed protocols such as WhatsApp [21, 33] (for client to server communication), Wiregurard [12, 13], Slack, Amazon AWS[1], and is potentially an ideal candidate for protecting the transport layer in IoT networks. Despite being distributed in applications used regularly by billions of users, there has not been a computational proof of Noise's security.[2]

Modularity in Cryptography and Real-World Definitions and analyses in cryptography usually aim to be as modular as possible such that the results are flexibly composable. In contrast, many real-world protocols are specifically designed for one purpose such that modularity – especially regarding single components of these protocols – is not necessary and maybe even undesired (e.g., due to worse performance).

For the generic (secure) composition of a key exchange protocol with a symmetric primitive (such as a symmetric channel), the key exchange protocol needs to provide key indistinguishability for the established symmetric key (among other properties; cf., Brzuska et al. [7]). However, if this symmetric key was used by the key exchange itself, it is not indistinguishable from a random key space element anymore (as an adversary can simply check whether the challenged key was used). The same property needs to hold, and the same obstacle arises for multi-stage key exchange protocols: in order to allow for generic compositions of key exchange and symmetric protocols, the symmetric key must not be used by the key exchange protocol itself in order to maintain modular composition.

Since many real-world protocols (such as TLS 1.2, Quic, Signal, Noise, TLS 1.3 and others) disregard modularity (in the sense that key exchange and channel are inextricably intertwined), cryptographic analyses of these protocols chose one out of the following three bypassing approaches: 1. pausing the protocol before

[1] Both Slack and AWS use it in internal server-to-server communication.
[2] Except for the single pattern that is employed in the Wireguard protocol [13, 28].

the key is internally used to prove key indistinguishability at that point (which still prevents generic composition results as the protocol uses the key afterwards), 2. analyzing a modified version of the protocol in which key exchange and channel are cleanly divided (which proves nothing about the actually used protocol), or 3. considering the security of the whole protocol instead of its single components by applying the ACCE model. In this work we follow the latter approach and – since no suitable ACCE model for staged protocols exist – propose a flexible and generalized ACCE model.

Flexibility and Generalization for ACCE Originally the Authenticated and Confidential Channel Establishment (ACCE) model was developed with the strict separation between key establishment and communication channel in mind. The security of ACCE, however, does not require this separation, because it only targets on the confidentiality of transmitted messages and the authentication among communicating parties.

Hence, our consideration of ACCE primitives differs from previous approaches that originated from notions of composition. We instead see fACCE as a primitive that is potentially built from authenticated key exchange (AKE) and secure channel protocols, but not necessarily *cleanly separated* into the "pre-accept" phase that establishes secrets and a "post-accept" phase that securely transmits payloads. We directly model all communication (handshake and payload transmission) via algorithms Enc and Dec which not only capture the secure channel but also *handshake operations* for the channel establishment. As the bytes sent over the network do not need to be further specified, we simply call them *ciphertexts* even though payload is not necessarily encrypted. We similarly view a *single* dedicated session key as a legacy of instantiating ACCE protocols via the composition of AKE and channel protocols. Since there are ways to secure the transmission of payload data other than simply using a symmetric key – consider asymmetric channels that use public key cryptography – we entirely subsume session-specific information in the session state. Furthermore, we drop length-hiding property [22] since we consider it not inherent in channel protocols.

After eliminating the boundary between handshake and channel, it is important to note that a protocol that establishes a channel immediately (i.e., with the first protocol message) cannot fulfill the same security guarantees as protocols that take multiple round-trips before allowing the confidential transmission of payload. This intuition can be compared to different security levels that are achieved by key encapsulation mechanisms (KEM), one-round-key exchanges (ORKE), and authenticated key exchanges (AKE) as depicted in Figure 1. For example, one message patterns (i.e., KEM-DEM constructions) are, among other deficiencies, subject to replay attacks if not equipped with expensive key update mechanisms such as in [19]. As a result, such attacks must be considered when designing an appropriate security model. Our model takes these different stages of security goals into account by adding flexibility to the ACCE notion.

As such we follow a similar approach as the multi-stage key exchange (MSKE) model. However, since our syntax allows for no distinction between stages (note that the MSKE model obtains new keys for each stage from the protocol), we

assume the considered protocols to output a *stage number* ς with every encryption and decryption. With ς, the protocol indicates the 'security level' of the transmitted message (e.g., towards an upper layer application). In the case of an ACCE protocol in which all security properties are reached at once, this stage number is equivalent to distinguishing between the pre- and post-accept phase. In case of multi-stage protocols, a security classification can be useful information for an upper layer application that can then decide when to transmit confidential content. Since there exists no other generic indication to differentiate multiple stages based on our syntax[3], it is essentially necessary for defining security (independent of a specific protocol) that the protocol itself outputs the stage numbers. Using these output stage numbers, one can specify for each stage which properties need to be reached by the protocol in order to achieve *security*. As a result, while one security property may not be reached in an early stage (and thus the adversary could trivially attack communication in this stage), later stages may reach this security property.

Further differences from the MSKE model are that we use a generic partnering notion (instead of protocol-dependent session identifiers), define authentication flexibly (e.g., unilateral authentication does not necessarily mean server authentication), provide a metric to meaningfully compare security statements of differing yet similar protocols, and, due to the ACCE nature of our model, provide valuable security statements on channels that are built using 'internal' symmetric keys (for which composition results of the MSKE models can naturally provide no generic guarantees).[4]

Contributions Our contributions can be summarized as follows:

- We generalize and flexibilize ACCE by finding its core idea and removing remnants of historic constructions and thereby propose a model to analyze channel establishment protocols with multiple stages, fulfilling different security properties. Though this model is due to its flexibility rather complex, we consider the overall generalizations useful for future analyses.
- We prove flexible ACCE security for the majority of Noise framework's standard protocol patterns in the full version of this article [15], considering multiple fine-grained security properties of patterns. By focusing on the security of the established channels instead of the established session keys, this allows us to comprehend security claims of the Noise specification. Here we give an intuition for our overall proof approach and depict the proof of pattern `XK` in full details.

[3] One could imagine that the round-trips in the protocol may serve as stages. However, one can only define round-trips in a protocol execution if both session participants can be observed (which is not the case when considering active adversaries).

[4] The composition theorems by Fischlin and Günther [14, 16, 17] explicitly exclude internally used keys (such that internal keys in Quic or TLS 1.3 cannot be used for generic symmetric primitives).

1.1 Related Work

Computational security proofs for real world protocols have a long history (e.g., [11, 13, 14, 16, 27, 30]). As described earlier, due to the usage of the channel key in the handshake of TLS 1.2, the ACCE model was introduced by Jager et al. [23] (which was later also used in [3, 6, 8, 9]) as a proof of key indistinguishability was impossible without considering a modified protocol variant. To further analyze the security of TLS 1.2 without client authentication, Krawczyk et al. [27] and Kohlar et al. [26] independently proposed a variant of the ACCE model.

The multi-stage key exchange (MSKE) model by Fischlin and Günther [16] extends the Bellare-Rogaway model [2] (further extended by [14, 17]) similarly as we extend the original ACCE model (by allowing protocols to reach different security properties at different stages during the execution). Due to the issue of key-usage during the handshake in Noise (as in TLS 1.2 or Signal) and further model restrictions, the multi-stage key exchange cannot be applied here.

Giesen et al. [18] extended the ACCE model to consider multiple stages during a protocol execution to analyze TLS renegotiation. Besides its static security definition(s) and in addition to inheriting other unnecessary remnants of the ACCE model, all stages necessarily consist of separate handshake and channel phases (making it unapplicable for generic multi-stage protocols). Another step towards considering stages in ACCE was taken by Lychev et al. [29] and more recently by Chen et al. [10]. Their QACCE and msACCE models are, however, strongly tailored to the respectively analyzed protocols (QUIC and TLS 1.3). Blazy et al. [4] also proposed very recently a multistage ACCE model to analyze a ratcheting protocol. Similarly, their model strongly depends on the analyzed protocol, pursuing a contrary strategy to ours (i.e., a specialized instead of a generic model).

Previous to our work, Dowling and Paterson [13] examined the WireGuard key exchange protocol [12], itself based upon a single variant of Noise called pattern `IKpsk2`. They show that analyzing WireGuard in a key-indistinguishability-based security framework is impossible, as the protocol relies on an encrypted message using the established session keys to act as a key-confirmation message. They instead modify the WireGuard key exchange protocol to morally capture the key confirmation message, and prove the modified construction secure. Recently Lipp et al. [28] confirmed the security of the WireGuard protocol by an automated analysis with CryptoVerif. Using this tool, they were able to produce a computational proof of security. Independently and concurrent to our work, Kobeissi et al. [24, 25] published a framework for the formal verification (and automatic code generation) of Noise patterns. In particular, they formalize Noise patterns and use transition logic to create symbolic models of dynamically chosen Noise patterns to allow automatic verification using ProVerif. This is a strong indication for Noise's security but the approach and the results can barely be compared with computational, reduction-based proofs with respect to generic security models. As their verification of all base Noise patterns is conducted automatically with respect to the security statements from the Noise specification and we provide a reduction-based proof of security in a general-

ized, flexible computational model manually, we see these two approaches to be complementary. We note that symbolic analyses disregard the actual representation of algorithms' input and output values. Thus, in symbolic analyses, cryptographic primitives are highly idealized. Consequently, while reduction-based proofs provide relations to well studied hardness assumptions, symbolic analyses assume "unconditional" security of these primitives. Nevertheless, automatic proofs are less error-prone and better scalable which enables Kobeissi et al. [25] to apply their analysis of even more security properties (e.g., multiple variants of forward-secrecy) on far more Noise patterns than our manual approach allows.

2 Preliminaries

Here we formalize the notation and provide intuitions for security assumptions that we will utilize in our analysis of the Noise Protocol Framework. Standard assumptions and security notions such as *collision resistance* for hash functions, security of *pseudo-random functions*, and further variants of the *PRF-Oracle-Diffie-Hellman* assumption can be found in the full version [15].

2.1 Notation

The following notation will be used throughout the paper. For $q \in \mathbb{N}$ by $[q]$ we denote the set $\{1, \cdots, q\}$. For a function $F : \{0,1\}^a \rightarrow \{0,1\}^b$, a describes the input length and b describes the output length of the function. If a or b take the "value" $*$ we say that the function is defined for inputs or outputs of arbitrary length. Let S be a finite set and let $|S|$ be its size. We say a value x is chosen uniformly at random by $x \leftarrow_{\$} S$. Let $\mathcal{A}$ be a probabilistic algorithm, we let $y \leftarrow_{\$} \mathcal{A}(x_1, ...)$ denote running $\mathcal{A}$ on input $(x_1, ...)$ with uniformly chosen random coins, and assigning the output to y. If $\mathcal{A}$ is a deterministic algorithm, then $y \leftarrow \mathcal{A}(x_1, ...)$ denotes that y is computed by $\mathcal{A}$ using $(x_1, ...)$ as input. By $y \leftarrow_{[r]} \mathcal{A}(x_1, ...)$ we denote that a probabilistic algorithm $\mathcal{A}$ is invoked deterministically by *consuming* its random coins from r (i.e., each random coin from r is used at most once). ϵ is the empty string and $\perp$ is a special element indicating no input or no output.

2.2 The PRF-Oracle-Diffie-Hellman Assumption

Here we give the symmetric variant of the generic PRF-ODH assumption, introduced by Dowling and Paterson [13]. Our modification additionally allows to capture a "dual-PRF" like assumption necessary for the Noise Protocol Framework. The basic PRF-ODH assumption was introduced Jager et al. [22] and discussed in detail by Brendel et al. [5].

Definition 1 (Dual generic PRF-ODH Assumption). *Let G be a cyclic group of order q with generator g. Let $\mathsf{PRF} : G \times \mathcal{M} \rightarrow \mathcal{K}$ be a function from a pseudo-random function family that takes a group element $k \in G$ and a salt*

value $m \in \mathcal{M}$ as input, and outputs a value $y \in \mathcal{K}$. We define a second PRF family $\mathsf{PRF}_d : \mathcal{M} \times G \to \mathcal{K}$, by setting $\mathsf{PRF}_d(m, e) = \mathsf{PRF}(e, m)$. We define a security notion, sym-lr-PRF-ODH *security, which is parameterised by: $\mathsf{l}, \mathsf{r} \in \{\mathsf{n}, \mathsf{s}, \mathsf{m}\}$ indicating how often the adversary is allowed to query "left" and "right" oracles (*ODHu *and* ODHv*), where* n *indicates that no query is allowed,* s *that a single query is allowed, and* m *that multiple queries are allowed to the respective oracle. Consider the following security game $\mathcal{G}^{\text{sym-lr-PRF-ODH}}_{\mathsf{PRF},G,p,\mathcal{A}}$ between a challenger $\mathcal{C}$ and adversary $\mathcal{A}$.*

1. *The challenger $\mathcal{C}$ samples $u, v \leftarrow_\$ \mathbb{Z}_p$ and provides g, g^u, g^v to the adversary $\mathcal{A}$.*
2. *If $\mathsf{l} = \mathsf{m}$, $\mathcal{A}$ can issue arbitrarily many queries to oracle* ODHu*, and if $\mathsf{r} = \mathsf{m}$ and $\mathsf{sym} = \mathsf{Y}$ to the oracle* ODHv*. These are implemented as follows:*
   - ODHu*: on a query of the form (S, x), the challenger first checks if $S \notin G$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{PRF}_\lambda(S^u, x)$ and returns y.*
   - ODHv*: on a query of the form (T, x), the challenger first checks if $T \notin G$ and returns $\perp$ if this is the case. Otherwise, it computes $y \leftarrow \mathsf{PRF}_\lambda(T^v, x)$ and returns y.*
3. *Eventually, $\mathcal{A}$ issues a challenge query x^*. It is required that, for all queries (S, x) to* ODHu *made previously, if $S = g^v$, then $x \neq x^*$. Likewise, it is required that, for all queries (T, x) to* ODHv *made previously, if $T = g^u$, then $x \neq x^*$. This is to prevent trivial wins by $\mathcal{A}$. $\mathcal{C}$ samples a bit $b \leftarrow_\$ \{0, 1\}$ uniformly at random, computes $y_0 = \mathsf{PRF}_\lambda(g^{uv}, x^*)$, and samples $y_1 \leftarrow_\$ \{0, 1\}^\lambda$ uniformly at random. The challenger returns y_b to $\mathcal{A}$.*
4. *Next, $\mathcal{A}$ may issue (arbitrarily interleaved) queries to oracles* ODHu *and* ODHv*. These are handled as follows:*
   - ODHu*: on a query of the form (S, x), the challenger first checks if $S \notin G$ or if $(S, x) = (g^v, x^*)$ and returns $\perp$ if either holds. Otherwise, it returns $y \leftarrow \mathsf{PRF}_\lambda(S^u, x)$.*
   - ODHv*: on a query of the form (T, x), the challenger first checks if $T \notin G$ or if $(T, x) = (g^u, x^*)$ and returns $\perp$ if either holds. Otherwise, it returns $y \leftarrow \mathsf{PRF}_\lambda(T^v, x)$.*
5. *At some point, $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$.*

We say that the adversary wins $\mathcal{G}^{\text{sym-lr-PRF-ODH}}_{\mathsf{PRF},G,p,\mathcal{A}}$ if $b' = b$ and define the advantage function

$$\mathsf{Adv}^{\text{sym-lr-PRF-ODH}}_{\mathsf{PRF},G,p,\mathcal{A}} = |2 \cdot \Pr[b' = b] - 1|.$$

We define the advantage of $\mathcal{A}$ in breaking the dual *security of* PRF-ODH *as:*

$$\mathsf{Adv}^{\text{d-PRF-ODH}}_{\mathsf{PRF},G,p,\mathcal{A}} = \max\left\{\mathsf{Adv}^{\text{sym-lr-PRF-ODH}}_{\mathsf{PRF},G,p,\mathcal{A}}, \mathsf{Adv}^{\text{sym-lr-PRF-ODH}}_{\mathsf{PRF}_d,G,p,\mathcal{A}}\right\}$$

Intuitively, the sym-lr-PRF-ODH assumption holds if the advantage $\mathsf{Adv}^{\text{sym-lr-PRF-ODH}}_{\mathsf{PRF},G,p,\mathcal{A}}$ of any PPT adversary $\mathcal{A}$ is negligible. For conciseness in the advantage statements, we omit the d-PRF-ODH, and instead use sym-lr to specify which PRF-ODH assumption we use. Further used variants of the assumption are in the full version [15].

3 The Noise Protocol Framework

The Noise Protocol Framework (hereafter referred to as "Noise") is a specification that describes a framework with which two party channel establishment protocols can easily be instantiated for multiple purposes. The core of the framework is represented by the definition of 15 base protocol patterns. Each of these patterns employs only four underlying cryptographic primitives: a Diffie-Hellman group, a hash function, a key derivation function, and an AEAD cipher. Depending on how these cryptographic primitives are combined, the channel establishment protocols achieve different cryptographic properties. The main properties (in addition to confidentiality) are: 1. Authentication and integrity, 2. Key compromise impersonation (KCI) resistance, 3. Forward-secrecy, and 4. Resistance against replay attacks. Another interesting security property that is achieved by the protocols, but not explicitly claimed, is: 5. Resistance against reveals of executions' random coins.

The 15 patterns mainly differ in the setup in which they can be deployed. There are patterns that do not require the initial distribution of users' long-term public keys (and either insist on the authentication of users by transmitting these keys either in plaintext or encrypted, or alternatively disregard authentication altogether), and patterns that are based on the previous distribution of users' public keys. The out-of-band mechanism for public-key distribution is outside the scope of the specification, but one can imagine scenarios in which these keys are manually configured, can be acquired from a trusted third party, or are shipped with the respective application that uses Noise.

While historic protocols strictly separated key establishment and channel, recent specifications (such as TLS 1.3) also allow these phases to be interleaved. This allows the early transmission of payload data but results in reduced – and perhaps staged – levels of security for this data. The Noise specification provides a detailed description of security properties for the data transmission in each round-trip of the handshake and for the channel of each pattern [32].

While a key feature of Noise is the omission of a negotiation of a pattern or the negotiation of the exact employed cryptographic algorithms (in contrast to TLS, Noise is intended to be used in settings in which all participants are configured equally), recent discussions on the mailing list consider negotiation as a feature in the future[5] – which we will not regard in our analysis. Also outside the scope of our analysis, Noise allows further features such as symmetric pre-shared keys.

Implementation Assumptions The Noise specification provides suggestions for some implementation details (but does not mandatorily require them). For our analysis, we assume that the protocol implementation follows these suggestions:

- No padding is employed (i.e., the length of the plaintext message is the same as the length of the encrypted message), and

[5] `https://moderncrypto.org/mail-archive/noise/2018/001495.html`

– If an algorithm is called irregularly (an initiator receives before sending once, a party waits for ciphertext but encryption is invoked, decryption fails, ...), then the respective algorithm outputs an empty state and aborts.

Furthermore, we do not consider the associated data input on sending and receiving payload after the handshake. As our syntax intentionally makes no difference between handshake and channel, we cannot consider this additional feature of the Noise channel, as it is not provided during the handshake. Finally, we assume the protocols to output information on the current level of security (which we explain in more details below).

3.1 Noise Protocol Patterns

Here we explain the details of Noise, necessary to understand the core protocols and their properties.

A pattern is defined by the knowledge of each participant regarding the long-term public key (or static public key) of the respective partner (before the handshake and during the handshake). For unidirectional patterns, the single letter of the pattern name indicates whether the initiator's long-term public key is not defined (`N`), trans(X)mitted during the handshake (`X`), or known by the receiver in advance (`K`). It is clear that, for unidirectional patterns, the receiver's long-term public key needs to be known by the initiator in advance since otherwise no payload can be encrypted to the receiver. In the two-letter names of interactive patterns, the first letter indicates whether the initiator's long-term public key is not defined, X-mitted, or known by the responder, and the second letter indicates the same for the responder towards the initiator. So in the `XK` pattern, the initiator knows the responder's long-term public key in advance and the responder obtains the initiator's long-term public key during the handshake. At the top of Figure 2 (in which we depict three example Noise patterns) it is denoted that the initiator knows the responder's long-term public key and the responder knows its long-term secret key for patterns `N` and `NK` a priori. For pattern `XK`, the initiator additionally knows its own key pair (of which the public key is sent to the responder during the protocol execution).

Finally, the Noise specification distinguishes whether the long-term public key is sent in plain or encrypted (for the former, the letter would be `I` instead of `X`). The specification defines all pairwise letter-combinations among the three variants `N`, `X`, `K`, the unidirectional patterns `N`, `X`, `K`, and the three variants in which the initiator sends its long-term (Identity) DH share in plaintext (i.e., `I_`).

At the left margin of Figure 1, we depict how the Noise patterns' algorithms are invoked for party **A** (matching our generic syntax, formally defined in Section 5).

The handshake of a Noise pattern always starts with the initialization of the local state *st* (via Init()). This local state contains:

1. ρ: a boolean that indicates the session's role (initiator/responder),
2. `pattern`: the pattern name,

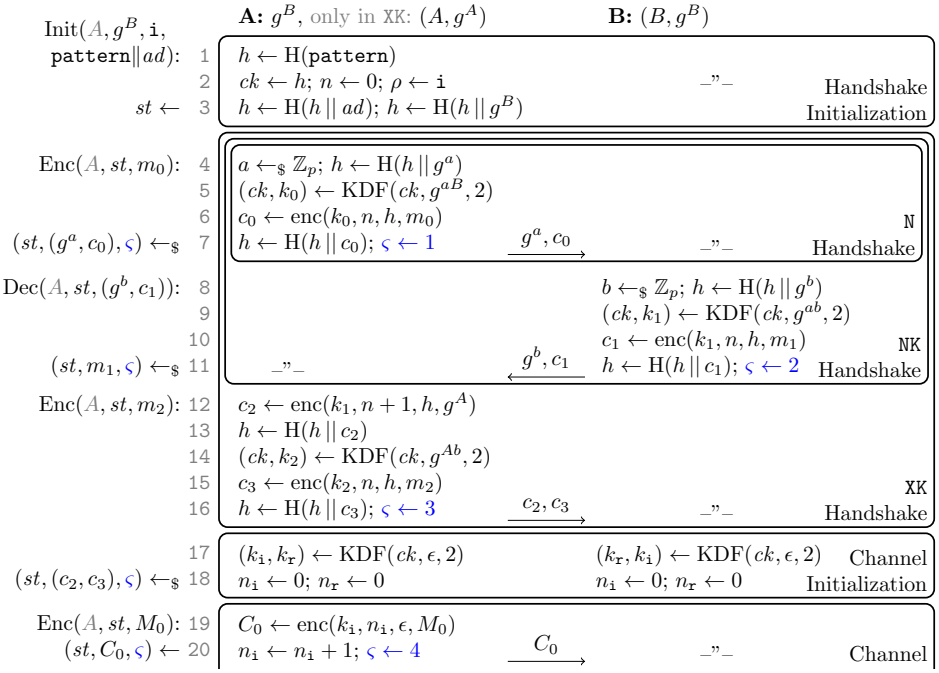

Fig. 2: Fully specified N,NK and XK patterns. m_i are payload messages sent *during* the handshake; M_i are payload messages sent *after* the handshake; ad is associated data with which the handshake is initiated; –"– denotes that the respective operations for receipt are processed (e.g., dec(c, ..) for $c \leftarrow$enc(..)). Handshake initialization, channel initialization, and channel are part of all patterns. Algorithm invocations and return values in the left column depict the interaction of party **A** with the protocol (showing the protocol's syntax defined in Section 5). Blue marked parts are not specified by Noise but are required for our analysis (thus we assume them to be part of the protocol).

3. (X, g^X): the session owner's long-term DH exponent and DH share (optional),
4. g^Y: the intended partner's long-term public DH share (optional),
5. (x, g^x): the session's ephemeral DH exponent and DH share (optional),
6. g^y: the peer's ephemeral public DH share (optional),
7. ck: the chaining key,
8. h: the hash variable,
9. k or $k_\mathtt{i}, k_\mathtt{r}$: the encryption key(s), and
10. n or $n_\mathtt{i}, n_\mathtt{r}$: the nonce(s) for encryption.

These values are set, considering the pattern name, associated data ad, and a priori known long-term public DH shares of the partners (see Figure 2 lines 1-4).

For each encryption (via Enc) during the handshake (i.e., before all desired security properties are reached), the following operations can be conducted:

(a) the generation of an ephemeral DH exponent and the transmission of the respective DH public share,
(b) the plain or encrypted transmission of a long-term DH share,
(c) the computation of a DH secret from a public DH share of the partner and their own DH exponent.

The actual operations in the protocol for operation (a) are 1. the sampling of a DH exponent, 2. the hashing of its public share into h, and 3. the transmission of this public share to the partner (lines 4,8). In case (b), the sender's long-term DH share is encrypted under the current key k and the resulting ciphertext is hashed into h and sent to the partner (lines 12-13).[6] If (c) a DH secret is computed, the current ck together with this DH secret are given as input to an invocation of the KDF (lines 5,9,14).

For each encryption during the handshake in which a key k was already computed, a ciphertext under this current key k is derived by encrypting a payload m or (if no payload exists yet) an empty string ϵ.[7] This ciphertext is sent to the partner and is also hashed into h. The current value of h is associated data for every encryption (lines 6-7,10-11,15-16).

After all handshake ciphertexts are processed, the channel is initialized with a symmetric key for each communication direction, derived by invoking the KDF on the current chaining key ck (lines 17-18). In one-message patterns such as `N`, payload can however only be sent from initiator to receiver.

Please note that we assume the protocol to additionally output information on the current payload transmission's level of security (represented by integer ς; lines 7,11,16,20). We proposed to add this feature to the Noise specification via the mailing list (as it could be useful to upper layer protocols). In Section 5, we describe why this feature is necessary for a security analysis in a generic model and in Section 6 we explain how this *stage counter* is naturally derived for the Noise patterns.

Flexibility in `N`, `NK`, `XK` Figure 2 depicts the three Noise patterns `N`, `NK`, and `XK`. As it can be seen, the `XK` pattern adds one further handshake ciphertext to the `NK` pattern such that the initiator is authenticated, and the `NK` pattern adds one handshake ciphertext to the `N` pattern, such that the responder is authenticated and a bidirectional forward secure channel is established.

4 Replay Attacks, State Reveals, and Their Relation

In our model, presented in Section 5, we allow adversaries to reveal the secret local states of session participants. Since this slightly raises the complexity of

[6] For patterns in which the long-term DH share is sent in plaintext, this DH share is directly hashed into h instead.

[7] Note that we use the algorithm Enc generically for sending information to the session partner. Confidentiality of payload is thereby not necessarily reached (see Section 5.1).

the model – as it induces a more careful treatment of *trivial* attacks – we contextualize the meaning and explain the importance of this adversarial power (in order to justify the increased complexity), and we give an intuition for relations between state reveals and replay attacks on a high level (in order support comprehensibility of the model). This is aimed to give some initial motivation on how we define the model formally in the next section.

Replay Attacks in Noise Replay attacks are an inevitable issue for early communication in many protocols – among them, many Noise patterns (e.g., patterns `N`, `NK`, and `XK`, cf., Figure 2). When assuming long-term keys of parties to be constant (and not variable; cf., [19]), the first ciphertext in a session, sent from an initiator instance π_i^s of a party i to a responder instance $\pi_j^{t_1}$ of a party j, can be replayed to all other (responder) instances $\pi_j^{t_2}, \ldots, \pi_j^{t_n}$ of party j. As long as instances of party j are not synchronized, they will not detect this replay attack (since the ciphertext is valid for all of them). Hence, they will all accept and process this ciphertext and reply with individual (valid) ciphertexts.

We observe three conditions that allow for replay attacks and that are true for seven out of the 15 standard patterns in Noise (cf., rows in Table 1 with $\texttt{rt} = 0$): (1) parties' long-term keys are static, (2) first ciphertexts in sessions from initiator to responder contain (confidential) payload already, and (3) there exists no (specified) synchronization mechanism among instances of a party. As a result, such ciphertexts in these patterns are inevitably potential subject to replay attacks.

Importance of State Reveal In general – independent of replay attacks – local states of instances contain crucial session secrets. Since the primitive that we consider in this work depicts not only the initialization of a session but the session itself (in contrast to, e.g., authenticated key exchange), these considered local secrets are stored and used until a session is terminated. For settings with long session duration (e.g., IoT networks), it is reasonable to assume that adversaries gain access to some instances' local secrets during the session lifetime. As a result, a realistic model should capture this adversarial power by allowing state reveals in the security game.

If state reveals were not allowed, protocols that store valuable secrets unnecessarily in the local state (e.g., own or partners' long-term secrets) would be declared secure even though this is intuitively insecure.

State Secrets under Replay Attacks In the following we describe an attack against an intuitively insecure protocol that would formally be declared "secure" in a model without state reveal. Consequently, we argue that such models are unsuitable for assessing *security*.

1. An initiator instance π_i^s of party i sends the first ciphertext c_0 in a session directed to a responder instance of party j, containing an ephemeral public encryption key pk^*, and stores the respective secret key sk^* in its local state. This ciphertext is not protected against replay attacks (for the reasons given above).

2. This ciphertext c_0 is forwarded to multiple responder instances $\pi_j^{t_1}, \ldots, \pi_j^{t_n}$ of party j.
3. Each responder instance $\pi_j^{t_l}, l \in [n]$ encrypts its individual, independent, confidential reply payload under pk^* in a ciphertext $\tilde{c}^l$. We note that this payload is encrypted forward-securely as (sk^*, pk^*) are independent of i's and j's long-term keys. Each instance $\pi_j^{t_l}$ sends a replay-resistant response c_1^l back to instance π_i^s, such that c_1^l contains $\tilde{c}^l$.
4. Instance π_i^s will only receive and process one reply $c_1^{t^*}$ from an instance $\pi_j^{t^*}$ and will continue a long-lived session with it. Instance $\pi_j^{t^*}$ encrypts all further payload to π_i^s under pk^* such that sk^* remains in π_i^s's local state until the session terminates.
5. All remaining involved sessions of party j will eventually terminate due to a timeout.
6. If an attacker obtains the local state of π_i^s before its session with $\pi_j^{t^*}$ terminates, it will learn all confidential replies, encrypted in ciphertexts $\tilde{c}^1, \ldots, \tilde{c}^n$ (in addition to the entire payload from $\pi_j^{t^*}$).

A model without state reveal declares all replies from instances $\pi_j^{t_1}, \ldots, \pi_j^{t_n}$ "secure" even though their security crucially relies on the secrecy of π_i^s's local state. Especially since countermeasures are trivial and highly efficient[8], we consider this example protocol insecure. This intuition matches the initial idea of the (key) reveal query in the Bellare-Rogaway model [2]: "Compromise of a session key should have minimal consequences" such that it should not "leak information about other (as yet uncompromised) session keys". Since we abstractly consider all local secrets of an instance combined as a generic state st, this condition should hold accordingly for its reveal. For protocols with early communication (allowing for replay attacks), this condition can, however, only be met as soon as a session has exactly two participants (i.e., the session continues with exactly one responder).

Depiction in the Model The essence of the above attack is not that messages are in danger due to replay attacks, but rather that secrets established during replay attacks may affect multiple instances. This effect must be reduced by secure protocols as far and as soon as possible. In our model, stage counters $\mathtt{rp}^{\mathtt{i}}, \mathtt{rp}^{\mathtt{r}}$ indicate how soon initiators and responders have local states that are independent of other instances' secrets (see Section 5.3). Thereby, we define reveals of their states harmless (for other instances) in case they are conducted after these stages are reached. Even though this independence may not be reached immediately (due to replay attacks in early communication), our model transparently indicates with these counters, how soon it is reached by protocols (see Table 1).

In our proof we explicitly emphasize the game hops in which it becomes clear that secrets are independent of others that are in revealable states of other instances.

[8] E.g., $\pi_j^{t_1}, \ldots, \pi_j^{t_n}$ encrypt individual random symmetric keys $k^1, \ldots, k^l$ under pk^* and encrypt their content under these symmetric keys respectively, then π_i^s can erase sk^* quickly such that its state is free of session-overlapping secrets.

5 Flexible ACCE Framework

The original ACCE model [22] and our generalization focus on the definition of *authentication* and *confidentiality* of messages, transmitted via a communication protocol (*channel establishment*). However in [22], traditional security goals like authentication and forward-secrecy are required to be reached before the actual channel is established.

Here we first provide a generic definition of the cryptographic primitive fACCE, then describe the standard execution environment in which its security is analyzed, further explain how we add flexibility to the adversary model with respect to the considered security properties, and define fACCE security.

5.1 fACCE Primitive Description

Below we define the *flexible ACCE* primitive. Intuitively, fACCE is a protocol that establishes a secure channel. Both the establishment of the channel and the transmission of payload through the channel are handled by the same algorithms. The special 'security level'-output ς of encryption and decryption signals which security properties are reached by the current algorithm invocation (e.g., to a higher level application).

Definition 2 (Flexible ACCE). *A flexible ACCE protocol* fACCE *is a tuple of algorithms* $\text{fACCE} = (\text{KGen}, \text{Init}, \text{Enc}, \text{Dec})$ *defined over a secret key space* $\mathcal{SK}$*, a public key space* $\mathcal{PK}$*, and a state space* $\mathcal{ST}$*. The syntax of an* fACCE *protocol is as follows:*

- $\text{KGen} \rightarrow_{\$} (sk, pk)$ *generates a long-term key pair where* $sk \in \mathcal{SK}, pk \in \mathcal{PK}$*.*
- $\text{Init}(sk, ppk, \rho, ad) \rightarrow_{\$} st$ *initializes a session to begin communication, where sk (optionally) is the caller's long-term secret key, ppk (optionally) is the long-term public key of the intended session partner,* $\rho \in \{\texttt{i}, \texttt{r}\}$ *is the session's role (i.e., initiator or responder), ad is data associated with this session, and* $sk \in \mathcal{SK} \cup \{\bot\}, ppk \in \mathcal{PK} \cup \{\bot\}, ad \in \{0,1\}^*, st \in \mathcal{ST}$*.*
- $\text{Enc}(sk, st, m) \rightarrow_{\$} (st', c, \varsigma)$ *continues the protocol execution in a session and takes message m to output new state* st'*, ciphertext c, and stage* ς *that indicates the security for the transmission via c of the input message, where* $sk \in \mathcal{SK} \cup \{\bot\}, st, st' \in \mathcal{ST}, m, c \in \{0,1\}^*, \varsigma \in \mathbb{N}$*.*
- $\text{Dec}(sk, st, c) \rightarrow_{\$} (st', m, \varsigma)$ *processes the protocol execution in a session triggered by c and outputs new state* st'*, message m, and stage* ς *that indicates the security for the output message during transmission via c, where* $sk \in \mathcal{SK} \cup \{\bot\}, st \in \mathcal{ST}, st' \in \mathcal{ST} \cup \{\bot\}, m, c \in \{0,1\}^*, \varsigma \in \mathbb{N}$*. If* $st' = \bot$ *is output, then this denotes a rejection of this ciphertext.*

We define as a convention that for output stage numbers $\varsigma = 0$, no security properties (in particular, no confidentiality) for the respectively transmitted payload has yet been reached.[9] Further we assume that the output stage numbers monotonically increase during a session (which is not a restriction).

[9] It is important to note that the first stage may not necessarily be 0, e.g. 0-RTT protocols that achieve confidentiality with the first message.

Please note that the syntax (and our security definition) leaves it to the specific protocol how far it enforces a ping-pong communication within a session. If the protocol automatically responds on received ciphertexts, we assume that the environment (in our security experiment this is depicted by the adversary) handles this.

We define the correctness of an fACCE protocol below. Intuitively an fACCE protocol is correct if messages, decrypted from the established channel, were equally sent to this channel by the partner.

Definition 3 (Correctness of fACCE**).** *An* fACCE *protocol is correct if, for any two key pairs* $(sk_{\mathtt{i}}, pk_{\mathtt{i}}), (sk_{\mathtt{r}}, pk_{\mathtt{r}})$ *output from* KGen *or set to* $(\perp, \perp)$ *respectively, their session states* $\text{Init}(sk_{\mathtt{i}}, pk_{\mathtt{r}}, \mathtt{i}, ad) \rightarrow_{\$} st_{\mathtt{i}}, \text{Init}(sk_{\mathtt{r}}, pk_{\mathtt{i}}, \mathtt{r}, ad) \rightarrow_{\$} st_{\mathtt{r}}$ *with* $ad \in \{0,1\}^*$*, and message-stage-ciphertext transcripts* $MSC_\rho, MSC_{\bar{\rho}} \leftarrow \epsilon$*, it holds for all sequences of operations* $((op^0, \rho^0, m^0), \ldots, (op^n, \rho^n, m^n))$ *(for all* $0 \leq l \leq n$ *with* $op^l \in \{e, d\}, \rho^l \in \{\mathtt{i}, \mathtt{r}\}, m^l \in \{0,1\}^*$*) that are executed as follows:*

- *if* $op^l = e$*, invoke* $\text{Enc}(sk_{\rho^l}, st_{\rho^l}, m^l) \rightarrow_{\$} (st_{\rho^l}, c^l, \varsigma^l)$ *and update* $MSC_\rho \leftarrow MSC_\rho \| (m^l, \varsigma^l, c^l)$*, or*
- *if* $op^l = d$*, invoke* $\text{Dec}(sk_{\rho^l}, st_{\rho^l}, c^l) \rightarrow_{\$} (st_{\rho^l}, m^l_*, \varsigma^l_*)$ *on* $(m^l_\circ, \varsigma^l_\circ, c^l) \| MSC_{\bar{\rho}} \leftarrow MSC_{\bar{\rho}}$ *and update it accordingly,*

that if $m^l_* \neq \perp$*, then encrypted and decrypted messages and stage outputs equal* $m^l_* = m^l_\circ, \varsigma^l_* = \varsigma^l_\circ$*, and that stage outputs increase monotonically (*$\forall l^* < l$ *with* $op^l = op^{l^*} = e$ *and* $\rho^{l^*} = \rho^l$ *it holds that* $\varsigma^{l^*} \leq \varsigma^l_\circ$*).*

5.2 Execution Environment

Here we describe the execution environment for our fACCE security experiment. In our model we allow the analysis of multiple security properties, and indeed allow these properties to be reached at different points during the protocol execution. As a consequence, one can specify for each stage which properties need to be reached by the protocol in order to achieve *security*. Since one security property may not be reached in an early stage (thus the adversary could trivially attack communication in this stage) and later stages may reach this security property, we need to separate the security experiment challenges that the adversary is to solve in each stage. We therefore define stage-specific challenge bits and freshness flags (opposed to one single challenge bit and a *static* freshness condition). The latter are dynamically checked and modified during the security game. We note that due to allowing secure and insecure stages within the same session, dependencies between messages may leak information to an attacker.

We consider a set of n_P parties each (potentially) maintaining a long-term key pair $\{(sk_1, pk_1), \ldots, (sk_{n_P}, pk_{n_P})\}$, $(sk_i, pk_i) \in \mathcal{SK} \times \mathcal{PK}$. In addition to the key pair, a variable $corr_i \in \{0,1\}$ is stored for every party $i \in [n_P]$ by the security experiment, that indicates whether sk_i was exposed to the adversary (via OCorrupt, see Section 5.4).

Each party can participate in up to n_S sessions. We denote both the set of variables that are specific for a session s of party i as well as the identifier of this

session as π_i^s. In addition to the local variables specific to each protocol, we list the set of per-session variables that we require for our model below. In order to derive or modify a variable x of session π we write $\pi.x$ to specify this variable.

- $\rho \in \{\mathtt{i}, \mathtt{r}\}$: The role of the session in the protocol execution (i.e., initiator or responder).
- $pid \in [n_P]$: The session partner's identifier.
- ad: Data associated with this session (provided as parameter at session initialization to Init).
- $T_e[\cdot], T_d[\cdot] \in \{0,1\}^*$: Arrays of sent or received ciphertexts. After every invocation of Enc or Dec of a session π_i^s, the respective ciphertext is appended to $\pi_i^s.T_e$ or $\pi_i^s.T_d$ respectively.
- $st \in \mathcal{ST}$: All protocol-specific local variables[10].
- $rand \in \{0,1\}^*$: Any random coins used by π_i^s's protocol execution.
- $(b_1, b_2, b_3, ...)$: A vector of challenge bits the adversary is to guess (one bit for each stage).
- $(fr_1, fr_2, fr_3, ...)$: A vector of freshness flags indicating whether the security of a stage in the session is considered to have been trivially broken by adversarial behavior.

At the beginning of the game, for all sessions π_i^s the following initial values are set: $\pi_i^s.T_e$, $\pi_i^s.T_d$, $\leftarrow \epsilon$, $\pi_i^s.fr_{\varsigma^*} \leftarrow 1$ for all $\varsigma^* \in \mathbb{N}$, and $\pi_i^s.rand \leftarrow_\$ \{0,1\}^*$, $\pi_i^s.b_{\varsigma^*} \leftarrow_\$ \{0,1\}$ for all $\varsigma^* \in \mathbb{N}$ are sampled.

Furthermore a set of ciphertexts $Rpl \leftarrow \emptyset$ is maintained in the security game, that are declared to initiate a non-fresh (replayed) session.

Partnering In order to define security in a flexible manner, we need to define partnering for sessions in the environment. Partnering is defined over the ciphertexts provided to/by the adversary via the oracles that let sessions encrypt and decrypt ($\mathsf{OEnc}, \mathsf{ODec}$). Intuitively, a session has an honest partner if everything that the honest partner received via ODec was sent by the session via OEnc (without modification) and vice versa, and at least one of the two parties received a ciphertext at least once[11]. This definition considers the asynchronous nature of the established channel, leading to a *matching conversation*-like partnering definition for fACCE.

Definition 4 (Honest Partner). *π_j^t is an honest partner of π_i^s if all initial variables match ($\pi_i^s.pid = j$, $\pi_j^t.pid = i$, $\pi_i^s.\rho \neq \pi_j^t.\rho$, $\pi_i^s.ad = \pi_j^t.ad$) and the received transcripts are a prefix of the partner's sent transcripts, respectively, where at least one them is not empty (i.e., for $a = |\pi_j^t.T_d|$, $b = |\pi_i^s.T_d|$ such that $a > 0$ if $\pi_i^s.\rho = \mathtt{i}$ and $b > 0$ if $\pi_i^s.\rho = \mathtt{r}$ then $\forall\ 0 \leq \alpha < a : (\pi_i^s.T_e[\alpha] = \pi_j^t.T_d[\alpha])$ and $\forall\ 0 \leq \beta < b : (\pi_i^s.T_d[\beta] = \pi_j^t.T_e[\beta])$). If π_i^s already received ciphertexts from π_j^t, then π_j^t is an honest partner of π_i^s only if there exists no other honest partner π^* of π_i^s (i.e., if $b > 0$ then there is no π^* such that π^* is an honest partner of π_i^s and $\pi^* \neq \pi_j^t$).*

[10] See Subsection 3.1 Noise's state definition.

[11] Note that this definition of *honest partnering* is symmetric (i.e., if a session π_i^s has an honest partner π_j^t, then this π_j^t has π_i^s as an honest partner as well).

Please note that after encrypting without decrypting yet, the initiator may have multiple honest partners (if the resulting ciphertexts are forwarded to multiple sessions). Due to the last requirement in Definition 4, our partnering notion requires that, after decrypting once, a session must have no more than one honest partner. Thereby partnering necessarily becomes a 1-to-1 relation as soon as the initiator received once from the responder.

5.3 Flexible Security Notion

Our model enables us to analyze *levels* of authentication and confidentiality – even for different stages within one protocol execution – and thereby to distinguish precisely if and when the following goals are reached: (a) Authentication and Integrity, (b) Forward-secrecy, and (c) Resistance against replay attacks. The extended version of this work [15] additionally considers KCI resistance and resistance against randomness reveal.

Our security definition, therefore, is indexed by five integers, called counters, ($\mathtt{au}^{\mathtt{i}}, \mathtt{au}^{\mathtt{r}}$, $\mathtt{fs}$, $\mathtt{rp}^{\mathtt{i}}, \mathtt{rp}^{\mathtt{r}}$) that indicate from which stage the respective property is achieved. Since properties can be established asymmetrically (e.g., a responder authenticates itself to an unauthenticated initiator in the first stage), some counters are indexed by role $\rho \in \{\mathtt{i}, \mathtt{r}\}$ (for initiator and responder respectively). One can think of each counter as a reference 'rung' on the 'ladder' of stages from which on the specified security property is achieved by the respectively analyzed protocol. Thus, as soon as the protocol outputs a certain stage that equals a counter (the protocol says that it reached the indicated 'rung' on the 'ladder'), all messages that are transmitted thereafter (including the message just encrypted or decrypted) reach the corresponding security property.[12] Please note that some security properties, such as authentication, develop their effect in two steps (see the trivial and real attacks in the description of oracle ODec in Section 5.4). We describe these counters below:

1. $\mathtt{au}^{\rho}$ defines the stage required for ρ to be authenticated. This means that it is hard to break the authenticity and integrity of ciphertexts *from* a party with role ρ (i.e., parties with role $\bar{\rho}$ reject ciphertexts if the origin is not an honest partner) if the stage number ς (output by Dec for the peer with $\bar{\rho}$) is greater or equal to $\mathtt{au}^{\rho}$. Note that, since our partnering notion considers FIFO channels, thereby also ciphertexts (and the order among them) are authenticated that were sent and received before $\mathtt{au}^{\rho}$ was reached.
2. $\mathtt{fs}$ defines the stage from which forward-secrecy (with respect to both session participants' long-term secrets) is reached. It is hard, for a stage $\varsigma \geq \mathtt{fs}$, to break the confidentiality of ciphertexts, even if both parties were corrupted.
3. $\mathtt{rp}^{\rho}$ defines the stage from which a fully revealed session state of ρ cannot be used to replay and reestablish the session. This means, for a session for which stage $\varsigma \geq \mathtt{rp}^{\rho}$ was reached, a revealed session state must not contain secrets

[12] Thereby only protocols are considered that monotonically increase security properties during sessions.

that affect the communication's security of any non-partnered sessions (especially of other receivers). The second condition of our partnering notion (cf., Definition 4) divides partners after replay attacks occur (i.e., marks them unpartnered thereby). Hence, protocols must diverge session state(s) of previous partners in case of such replay attacks, if partnering is used to control (and forbid) session state reveals. Only replayed first ciphertext(s) from an initiator to a responder do not to divide partners according to our partnering notion. In case of such session initiating replays, other sessions' states must be diverged by the protocol as soon as their stage has $\varsigma \geq \mathtt{rp}^{\rho}$.

We remark that our partnering notion already defines session participants unpartnered for all but one type of replay attacks: if ciphertexts, sent by an initiator that has already received a ciphertext once, or sent by a responder, are replayed, the respective receiver is defined to have no honest partner. In a security game in which state reveals are defined to be harmless for unpartnered sessions (which is the case for our model), this induces that such replay attacks force the protocol to diverge respective receivers' session states from their previous partners' session states. As a consequence, only replays of ciphertexts, sent by an initiator to (multiple) responder(s) without any reply from the latter, must be considered harmful in our security experiment. These replay attacks cannot be prevented if the receiver's long-term secret is defined static (which we do in contrast to e.g., [19]) and the initiator has never received a ciphertext. Our definition of replay attack resistance consequently focuses on the security damage that is caused by such replay attacks: it considers how soon the secrets, established by a (replayed) ciphertext, are independent among the sender and the (other) receivers of this replayed ciphertext. Hence, a session's secrets are recovered from a replay attack if they cannot be used to obtain information on other sessions' secrets.

Besides the explained prevention of replay attacks due to our partnering notion, ciphertexts that are transmitted before a stage $\varsigma > 0$ is output are (as also explained above) authenticated as soon as authentication is reached in a later stage. Apart from this, no security guarantees are required for ciphertexts transmitted under $\varsigma = 0$.

If a property is never reached in the specified protocol, then the respective counter is set to ∞ (e.g., for protocol with unauthenticated initiators, $\mathtt{au}^{\mathtt{i}} = \infty$).

5.4 Adversarial Model

In order to model active attacks in our environment, the security experiment provides the $\mathsf{OInit}, \mathsf{OEnc}, \mathsf{ODec}$ oracles to an adversary $\mathcal{A}$, who can use them to control communication among sessions, together with the oracles $\mathsf{OCorrupt}$, $\mathsf{OReveal}$.

Since our security definition becomes simpler and more clear by considering trivial attacks during the execution of the security game (not only as a separate freshness condition evaluated after the adversary terminated), we describe the

excluded trivial attacks and rewarded real attacks inline. The considered security properties are denoted as bullet point symbols below (in case they are not generically applicable).

The game maintains a win flag (to indicate whether the adversary broke authenticity or integrity of ciphertexts) and embeds challenge bits in the encryption (in order to model indistinguishability of ciphertexts). In order to win the security game, adversary $\mathcal{A}$ either has to trigger $\mathsf{win} \leftarrow 1$ or output the correct challenge bit $\pi_i^s.b_\varsigma$ of a specific session stage ς at the end of the game.

- $\mathsf{OInit}(i, s, j, \rho, ad)$ initializes a session π_i^s (if not yet initialized) of party i to be partnered with party j, invoking $\text{fACCE.Init}(sk_i, pk_j, \rho, ad) \rightarrow_{[\pi_i^s.rand]} \pi_i^s.st$ under $\pi_i^s.rand$. It also sets $\pi_i^s.\rho \leftarrow \rho$, $\pi_i^s.pid \leftarrow j$, and $\pi_i^s.ad \leftarrow ad$. This oracle provides no return value. Finally, the freshness flags are updated by invoking $\text{Fresh}_{\mathtt{fs}}()$ (see Figure 3).
- $\mathsf{OEnc}(i, s, m_0, m_1)$ triggers the encryption of message m_b for $b = \pi_i^s.b_\varsigma$ by invoking $\text{Enc}(sk_i, \pi_i^s.st, m_b) \rightarrow_{[\pi_i^s.rand]} (st', c, \varsigma)$ for an initialized π_i^s if $|m_0| = |m_1|$ and for $\varsigma = 0$ (i.e., confidentiality is not yet achieved) it must hold that $m_0 = m_1$ as the challenge bit would otherwise be trivially leaked. It updates the session specific variables $\pi_i^s.st \leftarrow st'$, returns (c, ς) to the adversary, and appends c to $\pi_i^s.T_e$ if $c \neq \perp$.
- $\mathsf{ODec}(i, s, c)$ triggers invocation of $\text{Dec}(sk_i, \pi_i^s.st, c) \rightarrow_{[\pi_i^s.rand]} (st', m, \varsigma)$ for an initialized π_i^s and returns (m, ς) if π_i^s has no honest partner, or returns ς otherwise (since challenges from the encryption oracle would otherwise be trivially leaked). Finally c is appended to $\pi_i^s.T_d$ if decryption succeeds.

  Excluding trivial attacks:

  - $\mathtt{fs}$: Since decryption can change the honesty of partners, the freshness flags are updated regarding corruptions by invoking $\text{Fresh}_{\mathtt{fs}}()$ (see Figure 3).
  - $\mathtt{au}$: The consideration of trivial attacks regarding authentication are a combination of the stage at which the protocol reaches authentication and corruptions of the participants' long-term secrets. If the received ciphertext was not sent by a session of the intended partner (i.e., there exists no honest partner) and
    1. party i is corrupted (i.e., $corr_i = 1$), then all following stages are marked un-fresh ($\pi_i^s.fr_{\varsigma^*} \leftarrow 0$ for all $\varsigma \leq \varsigma^*$), since this is a KCI attack.[13]
    2. neither party i nor the session's intended partner are corrupted (i.e., $corr_i = corr_{\pi_i^s.pid} = 0$) and authentication of the partner was not reached yet (i.e., $\varsigma < \mathtt{au}^{\pi_i^s.\bar{\rho}}$), then all following stages are marked un-fresh until authentication will be reached ($\pi_i^s.fr_{\varsigma^*} \leftarrow 0$ for all $\varsigma \leq \varsigma^* < \mathtt{au}^{\pi_i^s.\bar{\rho}}$), since this is a (temporary) trivial impersonation of the partner towards π_i^s.[14]

[13] Please note that resistance against KCI attacks is not required.

[14] If the partner authenticates later, then the protocol must ensure that this early trivial impersonation is detected. Consequently, this attack is not treated trivial anymore after the partner's authentication.

3. only the session's intended partner is corrupted (i.e., $corr_{\pi_i^s.pid} = 1 \neq corr_i$) and authentication of the partner was not reached yet or is reached with this received ciphertext (i.e., $\varsigma \leq \mathtt{au}^{\pi_i^s \cdot \bar{\rho}}$), then all following stages are marked un-fresh ($\pi_i^s.fr_{\varsigma^*} \leftarrow 0$ for all $\varsigma \leq \varsigma^*$), since this is (and will continue to be) a trivial impersonation of the partner towards π_i^s.

Rewarding real attacks:

`au`: Similarly to detecting trivial attacks, real attacks are rewarded by considering when authentication is reached in the respective protocol execution and if the participants' long-term secrets are corrupted.
The adversary breaks authentication (and thereby $\mathsf{win} \leftarrow 1$ is set) if the received ciphertext was not sent by a session of the intended partner but was successfully decrypted (i.e., there exists no honest partner and the output state is $st' \neq \bot$), the stage is still fresh ($\pi_i^s.fr_\varsigma = 1$), and
1. this is the first authenticated ciphertext ($\varsigma = \mathtt{au}^{\pi_i^s \cdot \bar{\rho}}$), and neither party i nor the intended partner are corrupted ($corr_i = corr_{\pi_i^s.pid} = 0$), or
2. this is a later authenticated ciphertext ($\varsigma > \mathtt{au}^{\pi_i^s \cdot \bar{\rho}}$) and party i is not corrupted ($corr_i = 0$) as this would otherwise be a KCI attack.

– $\mathsf{OCorrupt}(i) \rightarrow sk_i$ outputs the long-term secret key sk_i of party i, sets $corr_i \leftarrow 1$, and updates the freshness flags by invoking $\mathrm{Fresh}_{\mathtt{fs}}()$.

– $\mathsf{OReveal}(i, s) \rightarrow \pi_i^s.st$ outputs the current session state $\pi_i^s.st$.

Excluding trivial attacks:

- Revealing the session-state trivially determines this session's challenge bits, since the state contains any used session keys[15]. Hence $\pi_i^s.fr_{\varsigma^*} \leftarrow 0$ is set for all stages ς^*.
- Similarly, sufficient information is leaked to determine challenge bits embedded in ciphertexts to and from *all* honest partners π_j^t (and to impersonate π_i^s towards them). As such, $\pi_j^t.fr_{\varsigma^*} \leftarrow 0$ is set for all stages ς^* of these honest partners.

`rp`: In case the revealed secrets enable the adversary to obtain secrets of non-partnered sessions due to a replay attack ($\varsigma < \mathtt{rp}^{\pi_i^s \cdot \rho}$ where ς was output by π_i^s's last OEnc or ODec query) then the first ciphertext in this session is declared to induce non-fresh sessions via $Rpl \leftarrow Rpl \cup \{c\}$ where $c \leftarrow \pi_i^s.T_e[0]$ if $\pi_i^s.\rho = \mathtt{i}$ or $c \leftarrow \pi_i^s.T_d[0]$ if $\pi_i^s.\rho = \mathtt{r}$ (such that all sessions starting with this ciphertext are also marked non-fresh)[16].

[15] Since we do not consider forward-secrecy within sessions, the secret session state is considered to harm security of the whole session lifetime independent of when the state is revealed.

[16] One can easily define this trivial attack more specifically depending on whether this first ciphertext is authenticated and/or designated to a certain party. Depending on that, the secrets established by this ciphertext would only be valid among specific session (cf. [20]). For clarity and simplicity, we generically treat the ciphertext replayable solely. Please note that a state, revealed before the first ciphertext was sent/received (i.e., $c = \epsilon$), should not harm security of other sessions.

Freshness regarding Corruptions and Replays The definition of forward-secrecy, based on counter $\texttt{fs}$, is straight forward: if either the own long-term secrets or the intended partner's long-term secrets were corrupted (i.e., $corr_i = 1 \vee corr_{\pi_i^s.pid} = 1$), then only stages that provide forward-secrecy are marked fresh for the respective session (i.e., $\pi_i^s.fr_{\varsigma^*} \leftarrow 0$ for all $\varsigma^* < \texttt{fs}$). For sessions started with a ciphertext marked in set *Rpl* (i.e., initiating insecure communication due to the reveal of a replayable session), all stages are marked insecure. We formally define these properties via function $\text{Fresh}_{\texttt{fs}}()$ (see Figure 3).

$\text{Fresh}_{\texttt{fs}}()$:
For all $i \in [n_P]$, for all $s \in [n_S]$:
  $\texttt{ctr} \leftarrow \min(\varsigma^* : \pi_i^s.fr_{\varsigma^*} = 1)$
  If $corr_i = 1 \vee corr_{\pi_i^s.pid} = 1$:
    $\texttt{ctr} \leftarrow \max(\texttt{ctr}, \texttt{fs})$
  If $\pi_i^s.T_e[0] \in Rpl \wedge \pi_i^s.\rho = \texttt{i}$:
    $\texttt{ctr} \leftarrow \infty$
  If $\pi_i^s.T_d[0] \in Rpl \wedge \pi_i^s.\rho = \texttt{r}$:
    $\texttt{ctr} \leftarrow \infty$
  $\pi_i^s.fr_{\varsigma^*} \leftarrow 0$ for all $\varsigma^* < \texttt{ctr}$

Fig. 3: Function for updating freshness flags after each oracle invocation, considering long-term secrets' corruption (w.r.t. forward-secrecy) and full state reveals (w.r.t. replay attacks). The freshness flags up to (and excluding) the first secure stage are reset (e.g., for corrupted long-term keys, all stages in affected sessions are reset until forward-secrecy is reached).

5.5 Security Definition

The notion of fACCE security is captured as a game played by an adversary $\mathcal{A}$ in which the sessions are implemented as described above. At the beginning of the game, n_P long-term key pairs (pk_i, sk_i) $\forall i \in [n_P]$ are generated via fACCE.KGen and the respective public keys are provided to $\mathcal{A}$ as a parameter on the invocation (i.e., the start of the game). $\mathcal{A}$ interacts with the game via the queries described above and eventually terminates, potentially outputting a tuple (i, s, ς, b').

We can now turn to defining (in-)security of a fACCE protocol.

Definition 5 (Advantage in Breaking Flexible ACCE). *An adversary $\mathcal{A}$ breaks a flexible ACCE protocol* fACCE *with authentication stages* $(\texttt{au}^{\texttt{i}}, \texttt{au}^{\texttt{r}})$*, forward-secrecy stage* $\texttt{fs}$*, and replayability resistance stages* $(\texttt{rp}^{\texttt{i}}, \texttt{rp}^{\texttt{r}})$*, when $\mathcal{A}$ terminates and outputs (i, s, ς, b'), if there either exists a session π_i^s such that $\pi_i^s.b_\varsigma = b'$, and $\pi_i^s.fr_\varsigma = 1$ (which we subsume as event* guess*), or* $\mathsf{win} = 1$*. We define the advantage of an adversary $\mathcal{A}$ breaking a flexible ACCE protocol* fACCE *as* $\mathsf{Adv}_{\mathcal{A}}^{\text{fACCE}} = (2 \cdot \Pr[\mathsf{guess}] - 1) + \Pr[\mathsf{win} = 1]$.

Intuitively, a fACCE protocol is secure if it is correct and $\mathsf{Adv}_{\mathcal{A}}^{\text{fACCE}}$ is negligible for all probabilistic algorithms $\mathcal{A}$ running in polynomial-time.

Necessity of Holistic Model Our definition of flexible ACCE considers multiple security properties simultaneously (as opposed to having separate definitions for each regarded security property). In order to reduce complexity, it could seem useful to regard the security properties independently and then assemble the results. In the full version [15, Appendix B] we explain why this approach would produce more complexity, less comprehensibility, and is partially impossible.

6 Protocol Analyses

In this section, we provide an overview of our results of analyzing the Noise Protocol framework in our new fACCE model. Our main contribution is the full proofs of Noise Patterns `N`, `NN`, `NX`, `NK`, and `X`, `XN`, `XX`, `XK`. We focus on proving these two protocol "families" to demonstrate how our analysis can capture the wide variety of security properties that we show in Table 1, while also simplifying our approach by the re-use of our proof strategies. We give a detailed look of the proofs of Noise Pattern `XK` here and extend these proofs, considering further security properties in the full model, together with the proofs for the remaining mentioned patterns in the full version [15]. We present the analysis of Noise Pattern `XK` here as it comprehensibly provides an idea of the general proof structure and shows how Noise patterns can be built upon another. As the handshake of `XK` extends `NK`'s handshake, which in turn extends the handshake of `N` by a half round-trip respectively, each extension also results in further security properties (see Figure 2 and Table 1).

Generic Proof Structure The modular design of the Noise Protocol Framework allow us to write proofs that have a reasonably generic structure. While the proof for each specific Noise Pattern is distinct, each proof is, on a high level, split into two cases:

- The adversary has forged a ciphertext successfully, and sent it to a session that does *not* detect the forgery (or abort the protocol run). This case may be further split into multiple cases depending on which ciphertext in the Noise Pattern the adversary has managed to forge.
- The adversary has guessed the challenge bit correctly when it terminates the experiment.

We determine which $\mathsf{OCorrupt}$ queries cannot have been issued such that the attacked stage is still 'fresh' (as the adversary would otherwise be unsuccessful). Thus, each case has some queries that have not been issued to the session π_i^s and its partner session (where π_i^s either accepted the forged ciphertext, or the adversary output (i, s, ς, b')). In both cases we use a tailored $\mathsf{PRF\text{-}ODH}$ assumption, depending on which pair of queries (targeting long-term DH shares, state secrets, or, in the full model, ephemeral DH shares that depend on random coins) have *not* been issued, to replace the appropriate Diffie-Hellman public values and shared Diffie-Hellman secrets (using the ODH oracles to compute any additional secrets using the DH secret keys, if necessary). Afterwards, we iteratively

replace intermediate secrets derived during the protocol execution using PRF assumptions on the underlying key derivation function. Finally, we use a single (or potentially series of) AEAD assumption(s) to replace the encryptions of ciphertexts sent to, and decryption of ciphertexts arriving at, the session π_i^s. Any adversary capable of distinguishing these changes is able to break one of the underlying assumptions used, and depending on which case we are in, either: 1. The adversary is unable to forge a ciphertext to the session π_i^s, or 2. The adversary is unable to guess the challenge bit b with non-negligible probability.

This (high-level) description effectively captures the strategy we use to prove our statements about the Noise Patterns that we analyze.

Mapping Noise's Security Statements to our Model's Counters Here we define the exact modeled security via the stage counters $(\mathtt{au}^{\mathtt{i}}, \mathtt{au}^{\mathtt{r}}, \mathtt{fs}, \mathtt{rp}^{\mathtt{i}}, \mathtt{rp}^{\mathtt{r}})$, $(\mathtt{kc}^{\mathtt{i}}, \mathtt{kc}^{\mathtt{r}}, \mathtt{eck}, \mathtt{rl}^{\mathtt{i}}, \mathtt{rl}^{\mathtt{r}})$[17], used in our theorems of each proof. We also explain how they relate to the round-trips in the protocol execution of the respective Noise pattern (we discuss generic mapping among stage counters and round-trips in the full version [15, Appendix C.3]). For each of the base patterns of the Noise specification, the stage at which the respective security property is reached is listed in Table 1. As stage numbers ς output by the $\mathsf{Enc}, \mathsf{Dec}$ algorithms are defined as integers, we assume the Noise patterns to output a counter as stage number with every algorithm invocation, starting by 1 and always incremented by 1 until no further security properties are reached. In the case that the initiator's first ciphertext provides no confidentiality, the stage output is 0 (see column $rt = 0.5$ in Table 1) but the reply by the responder continues with $\varsigma = 2$.

The counters/round-trips for authentication and KCI resistance $(\mathtt{au}^{\rho}, \mathtt{kc}^{\rho})$ are directly lifted from the Noise specification [32]. As the definition of the remaining security properties deviate from the specification (or are not specified therein), the theorems' stage counters are defined as the first round-trips and stages that achieve the respective goals. Regarding forward-secrecy, the Noise specification differentiates among role dependent weak and strong variants of long-term secrets' corruptions. However, our consideration of forward-secrecy focuses on the relation between corruptions of long-term secrets and the reveal of sessions' random coins. Consequently, the counter $\mathtt{fs}$ is only partially derived from the Noise specification.

Resistance against replay attacks in the Noise specification only considers the adversary's ability to successfully let multiple sessions receive the same sent ciphertext. However, local state variables (like an ephemeral symmetric encryption key or a DH exponent), established by a ciphertext, can be exploited by an adversary to attack other sessions that sent or received the same (replayed) ciphertext. Such state variables may stay in the local state even after the replay attack "is over" (i.e., after only a unique honest partner exist). As the adversary is allowed to reveal the local state, our definition of replay attack resistance goes beyond others in the literature (e.g., [17]) and the Noise specification: it says that resistance against replay attacks is reached if the local state of a session

[17] The latter are only relevant for the proofs in the full model.

is independent of any other session's state (except from the respective unique honest partner).

	rt	$\mathtt{au}^{\mathtt{i}}$	$\mathtt{au}^{\mathtt{r}}$	fs	$\mathtt{rp}^{\mathtt{i}}$	$\mathtt{rp}^{\mathtt{r}}$	$\mathtt{kc}^{\mathtt{i}}$	$\mathtt{kc}^{\mathtt{r}}$	eck	$\mathtt{rl}^{\mathtt{i}}$	$\mathtt{rl}^{\mathtt{r}}$
N*	0	∞	∞	∞	∞	∞	∞	∞	∞	1	∞
X*	0	1	∞	∞	∞	∞	∞	∞	1	1	∞
K	0	1	∞	∞	∞	∞	∞	∞	1	1	∞
NN*	0.5	∞	∞	2	2	0	∞	∞	∞	∞	∞
NK*	0	∞	2	2	2	2	∞	2	∞	1	∞
NX*	0.5	∞	2	2	2	0	∞	2	∞	2	∞
XN*	0.5	3	∞	2	2	0	3	∞	∞	∞	3
XK*	0	3	2	2	2	2	3	2	∞	1	3
XX*	0.5	3	2	2	2	0	3	2	∞	2	3
KN	0.5	3	∞	2	2	0	3	∞	∞	∞	2
KK	0	1	2	2	2	2	3	2	1	1	2
KX	0.5	3	2	2	2	0	3	2	∞	2	2
IN	0.5	3	∞	2	2	0	3	∞	∞	∞	2
IK	0	1	2	2	2	2	3	2	1	1	2
IX	0.5	3	2	2	2	0	3	2	∞	2	2

Table 1: Stages at which the respective security properties are reached. Stage x is reached (and thus returned by the protocol via output ς) at round-trip $\mathrm{RT}(x) = x/2$ (for $\mathrm{RT}(x) < rt$ no property is reached). The right half of columns depicts the counters for security properties that are only considered in the full model. $\mathtt{au}^{\rho}, \mathtt{kc}^{\rho}$ were extracted from Noise's specification [32]; $\mathtt{fs}, \mathtt{rp}^{\rho}$ are related to their definition in the specification (but adapted to our model). $\mathtt{rl}^{\rho}, \mathtt{eck}$ were defined purely with respect to the model. We give proofs for the patterns marked with a *.

6.1 Proof of Noise Pattern XK

Theorem 1. *Noise protocol* XK *(as in Figure 2) is an fACCE-secure protocol with authentication levels* $\mathtt{au} = (3, 2)$*, forward-secrecy* $\mathtt{fs} = 2$*, and replay resistance* $\mathtt{rp} = (2, 2)$*. For an adversary* $\mathcal{A}$ *against the flexible ACCE security game (defined in Section 5) one can efficiently define adversaries* $\mathcal{B}_{\mathsf{coll}}$ *against the collision resistance of* H*,* $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ *against the PRF-ODH assumptions* ms-PRF-ODH*,* sn-PRF-ODH *and* sym-ms-PRF-ODH *with respect to group* G *and* KDF*,* $\mathcal{B}_{\mathsf{aead}}$ *against the AEAD security of* AEAD*, and* $\mathcal{B}_{\mathsf{prf}}$ *against the PRF security of* KDF *with:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{fACCE}}_{\mathtt{XK},n_P,n_S,\mathcal{A}} \leq\ & 3 \cdot \mathsf{Adv}^{\mathsf{coll}}_{\mathrm{H},\mathcal{B}_{\mathsf{coll}}} + n_P^2 n_S \cdot \big(\mathsf{Adv}^{\mathsf{prf}}_{\mathrm{KDF},\mathcal{B}_{\mathsf{prf}}} + \mathsf{Adv}^{\mathsf{ms\text{-}PRF\text{-}ODH}}_{\mathrm{KDF},G,p,\mathcal{B}_{\mathsf{PRF\text{-}ODH}}} \\ & + \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{AEAD},\mathcal{B}_{\mathsf{aead}}}\big) + n_P^2 n_S^2 \cdot \big(\mathsf{Adv}^{\mathsf{aead}}_{\mathsf{AEAD},\mathcal{B}_{\mathsf{aead}}} + \mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRF\text{-}ODH}}_{\mathrm{KDF},G,p,\mathcal{B}_{\mathsf{PRF\text{-}ODH}}}\big) \\ & + n_P^2 n_S^2 \cdot \Big(\mathsf{max}\Big\{\big(3 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathrm{KDF},\mathcal{B}_{\mathsf{prf}}} + \mathsf{Adv}^{\mathsf{ms\text{-}PRF\text{-}ODH}}_{\mathrm{KDF},G,p,\mathcal{B}_{\mathsf{PRF\text{-}ODH}}} + 4 \cdot \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{AEAD},\mathcal{B}_{\mathsf{aead}}}\big), \\ & \big(2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathrm{KDF},\mathcal{B}_{\mathsf{prf}}} + 3 \cdot \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{AEAD},\mathcal{B}_{\mathsf{aead}}} + \mathsf{Adv}^{\mathsf{sn\text{-}PRF\text{-}ODH}}_{\mathrm{KDF},G,p,\mathcal{B}_{\mathsf{PRF\text{-}ODH}}}\big)\Big\}\Big).\end{aligned}$$

Proof. We give below the proof of Noise Pattern XK. We split our analysis into three cases, depending on *how* the adversary can win the experiment. For the first two cases, the adversary causes $\mathsf{win} \leftarrow 1$ if the received ciphertext was not sent by a session of the intended partner, but was successfully decrypted in an authenticated stage by either an initiator (**Case A**) or a responder (**Case B**) session. For **Case A** $\mathcal{A}$ cannot have issued a $\mathsf{OCorrupt}(\pi_i^s.pid)$ query because breaking authentication of a corrupted peer is a trivial attack (as $\mathtt{au}^{\mathtt{r}} = 2$). Similiarly, for **Case B** $\mathcal{A}$ cannot have issued a $\mathsf{OCorrupt}(\pi_i^s.pid)$ query as $\mathtt{au}^{\mathtt{i}} = 3$). Next we focus on an adversary attempting to guess the challenge bit b for any fresh session (**Case C**). **Case C** is further separated into two subcases, depending on the combination of allowable $\mathsf{OCorrupt}$ queries $\mathcal{A}$ issues, as defined in Section 5. We show that under such restrictions, $\mathcal{A}$ has a negligible advantage in guessing a challenge bit b for the session π_i^s. We begin with the standard fACCE experiment defined in Section 5, and treat **Case A**.

In **Case A Game 1**, we define an abort event that triggers if a hash collision occurs. We do so by defining an algorithm $\mathcal{B}_{\mathsf{coll}}$ that computes all hash values honestly, and aborts if there exist two evaluations $(in, \mathrm{H}(in)), (\hat{in}, \mathrm{H}(\hat{in}))$ such that $in \neq \hat{in}$, but $\mathrm{H}(in) = \mathrm{H}(\hat{in})$, outputting this pair to a hash collision challenger if found. In the next two games (**Game 2**, **Game 3**) we guess the index (i, s) of the session π_i^s, as well as the index j of the honest partner π_j^t and abort if either $\mathcal{A}$ terminates and outputs $(i^*, s^*, \varsigma, b')$ such that $(i^*, s^*) \neq (i, s)$, or if $\mathcal{A}$ initialises π_i^s such that $\pi_i^s.pid \neq j$. From now, the challenger playing the fACCE game "knows" at the beginning of the experiment the index of the session that $\mathcal{A}$ will target, and its intended partner j. In **Game 4**, we introduce an abort event $abort_{win}$ that triggers if the challenger sets $\mathsf{win} \leftarrow 1$ when the test session proceses the ciphertext (g^b, c_1). The rest of the game hops in **Case A** now bound the advantage of $\mathcal{A}$ in causing $abort_{win}$ to occur.

Case A Game 5 requires careful consideration: Note that by **Game 2**, we know at the beginning of the experiment the index of session π_i^s such that (i, s, ς', b') is output by the adversary. Similarly, by **Game 3**, we know at the beginning of the experiment the index of the intended partner $\pi_i^s.pid$ of the session π_i^s. Thus, we define an algorithm $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ that initializes a $\mathsf{ms\text{-}PRF\text{-}ODH}$ challenger, embeds the DH challenge keyshare g^u into the long-term public-key of party j, embeds the DH challenge keyshare g^v into the ephemeral public-key of session π_i^s, replaces the computation of $ck, k_0 \leftarrow \mathrm{KDF}(ck, g^{aB}, 2)$ (in the session π_i^s and its partner) with uniformly random values $\widetilde{ck}, \widetilde{k_0}$, and gives $pk_j = g^u$ to the adversary with all other (honestly generated) public keys. However, $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ must account for all sessions t such that party j must use the private key for computations. In the Noise Protocol XK, the long-term private keys are used in the following ways: In sessions where the party j acts as the initiator, they compute $ck, k_2 \leftarrow \mathrm{KDF}(ck, g^{xu}, 2)$. Similarly, in sessions where the party acts as the responder, they compute $ck, k_0 \leftarrow \mathrm{KDF}(ck, g^{xu}, 2)$. To simulate this computation, $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ must instead use the ODHu oracle provided by the $\mathsf{ms\text{-}PRF\text{-}ODH}$ challenger, specifically querying $\mathsf{ODHu}(ck, X)$, (where X is the Diffie-Hellman public keyshare such that the private key is unknown to the challenger)

which will output $\mathrm{KDF}(ck, X^u)$. We note that $\mathtt{au}^{\mathtt{r}} = 2$, and only after processing (g^b, c_1) will π_i^s output $\varsigma = 2$, and so $\mathcal{A}$ cannot issue a $\mathsf{OCorrupt}(j)$ query before π_i^s processes ciphertext g^b, c_1. Thus we bound the probability of $\mathcal{A}$ distinguishing this change by the security of the ms-PRF-ODH assumption.

In **Case A Game 6** the challenger replaces the concretely computed values $ck, k_1 \leftarrow \mathrm{KDF}(\widetilde{ck}, g^{ab}, 2)$ in π_i^s and its honest partner (if one exists), with uniformly random values $\widetilde{ck}, \widetilde{k_1}$. As by **Game 5**, the input $\widetilde{ck}$ is already uniformly random and independent of the protocol execution, distinguishing this game hop can be reduced to the prf security of the KDF. Note that due to this change, the state of π_i^s (containing only $\widetilde{ck}, \widetilde{k_1}$ as secrets) is independent of other sessions (making it useless to reveal their states; cf., counters $\mathtt{rp}^{\mathtt{i}} = \mathtt{rp}^{\mathtt{r}} = 2$).

Case A Game 7 proceeds identically to **Game 6**, except that the challenger flips a bit $\bar{b}$, and uses $\bar{b}$ instead of $\pi_i^s.b_1$ when responding to OEnc or ODec queries from $\mathcal{A}$ directed to sessions π_i^s or π_j^t when using the key $\widetilde{k_1}$. We do so by constructing an algorithm $\mathcal{B}_{\mathsf{aead}}$ that interacts with an aead challenger, and forwards such OEnc or ODec queries to the aead challenger. This change reduces to the aead security of the AEAD scheme, and since $\widetilde{k_1}$ is a uniformly random and independent value by **Game 6**, this replacement is sound. The additional-data field of c_1 contains $h = \mathrm{H}(\mathrm{H}(\mathrm{H}(\mathrm{H}(\mathrm{H}(\mathtt{XK\_label} \| ad) \| g^B) \| g^a) \| c_0) \| g^b)$. By **Game 1** we abort the experiment if $\mathcal{A}$ causes a hash-collision to occur, and by **Game 4** we abort if no honest session owned by j has output g^b, c_1. An adversary capable of causing $\mathsf{win} \leftarrow 1$ when π_i^s processes the ciphertext g^b, c_1 can break the aead security of the underlying AEAD scheme, and thus $\mathcal{A}$ has no advantage in causing $abort_{win}$ to occur.

$$\mathsf{Adv}^{\mathsf{fACCE},\ \mathbf{Case\ A}}_{\mathtt{XK},n_P,n_S,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{coll}}_{\mathrm{H},\mathcal{B}_{\mathsf{coll}}} + n_P^2 n_S \cdot \Big(\mathsf{Adv}^{\mathsf{ms\text{-}PRF\text{-}ODH}}_{\mathrm{KDF},G,p,\mathcal{B}_{\mathsf{PRF\text{-}ODH}}} + \mathsf{Adv}^{\mathsf{prf}}_{\mathrm{KDF},\mathcal{B}_{\mathsf{prf}}} + \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{AEAD},\mathcal{B}_{\mathsf{aead}}} + \mathsf{Adv}^{\mathsf{ms\text{-}PRF\text{-}ODH}}_{\mathrm{KDF},G,p,\mathcal{B}_{\mathsf{PRF\text{-}ODH}}}\Big)$$

We can now treat **Case B**.

The first four games (**Game 1**, **2**, **3**, **4**) proceed similarly to **Case A**. That is, we abort when a hash-collision is detected, and guess the index (i, s) of the first session π_i^s to set $\mathsf{win} \leftarrow 1$. However, in **Game 3**, we additionally guess the index (j, t) of the intended partner *session*, and abort if our guess is incorrect. **Game 4** still introduces an abort event that occurs if $\mathsf{win} \leftarrow 1$ is set in the test session, and the rest of the game hops bound the advantage of $\mathcal{A}$ in causing the abort event to occur.

Case B Game 5 again requires careful consideration: Note that by **Game 2**, we know at the beginning of the experiment the index of session π_i^s such that (i, s, ς', b') is output by the adversary and by **Game 3**, we know at the beginning of the experiment the index of the honest partner session (j, t) of the session π_i^s. We take a similar approach to **Game 5** of **Case A**. However, in this game we replace the computation of $ck, k_2 \leftarrow \mathrm{KDF}(ck, g^{Ab}, 2)$ with uniformly random and independent values $(\widetilde{ck}, \widetilde{k_2})$ in the test session and its honest partner. Specifically, we define an algorithm $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ that initialises a sym-ms-PRF-ODH challenger, embeds the DH challenge keyshares g^u into the long-term public-key

of party i, embeds the DH challenge keyshare g^v into the ephemeral public-key of session π_j^t, replaces the computation of $ck, k_2 \leftarrow \text{KDF}(ck, g^{Ab}, 2)$ (in the session π_i^s and its partner) with uniformly random values $\widetilde{ck}, \widetilde{k_2}$, and gives $pk_i = g^u$ to the adversary with all other (honestly generated) public keys. However, $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ must account for all sessions s such that party i must use the private key for computations. In the Noise Protocol `XK`, the long-term private keys are used in the following ways: In sessions where the party i acts as the initiator, they compute $ck, k_2 \leftarrow \text{KDF}(ck, g^{xu}, 2)$. Similarly, in sessions where the party acts as the responder, they compute $ck, k_0 \leftarrow \text{KDF}(ck, g^{xu}, 2)$. To simulate this computation, $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ must instead use the ODHu oracle provided by the $\mathsf{ms\text{-}PRF\text{-}ODH}$ challenger, specifically querying $\mathsf{ODHu}(ck, X)$, (where X is the Diffie-Hellman public keyshare such that the private key is unknown to the challenger) which will output $\text{KDF}(ck, X^u)$. However, $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ must account for the fact that the private key of g^v (the ephemeral public-key of π_i^s) is actually used before the computation of ck, k_2. In particular, it is used earlier in the protocol to compute $ck, k_0 := \text{KDF}(ck, g^{av})$, where g^a may have been contributed by $\mathcal{A}$. In this case, in order to compute ck, k_0, $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ must instead use the ODHv oracle provided by the $\mathsf{sym\text{-}ms\text{-}PRF\text{-}ODH}$ challenger, specifically querying $\mathsf{ODHv}(ck, g^a)$, which will output $\text{KDF}(ck, g^{av})$. We note that $\mathtt{au}^{\mathtt{i}} = 3$, and only after processing (c_2, c_3) will π_i^s output $\varsigma = 3$, and so $\mathcal{A}$ cannot issue a $\mathsf{OCorrupt}(i)$ query before π_i^s processes ciphertext c_2, c_3. Thus we bound the probability of $\mathcal{A}$ distinguishing this change by the security of the $\mathsf{sym\text{-}ms\text{-}PRF\text{-}ODH}$ assumption. Note that other session states are (and were) independent of π_i^s's state as g^A is not stored in a state, a collision with g^b would break the above game hop, and $\widetilde{ck}, \widetilde{k_2}$ were randomly sampled (cf., counters $\mathtt{rp}^{\mathtt{i}}, \mathtt{rp}^{\mathtt{r}}$). **Case B Game 6** proceeds identically to **Game 5**, except that the challenger responds to OEnc or ODec queries directed to π_i^s or π_j^t outputting $\varsigma = 3$ from $\mathcal{A}$ (i.e. when using the key $\widetilde{k_2}$) and aborts if π_i^s decrypts c_2, c_3 successfully, but it was not output by an honest partner. This changes reduces to the AEAD security of the AEAD scheme. The additional-data field of c_3 contains $h = \text{H}(\text{H}(\text{H}(\text{H}(\text{H}(\text{H}(\mathtt{XK\_label} \| ad) \| g^B) \| g^a) \| c_0) \| g^b) \| c_1) \| c_2)$. By **Game 1** we abort the experiment if $\mathcal{A}$ causes a hash-collision to occur, and by **Game 4** we abort if no honest session owned by j has output c_2, c_3. Now, $\mathcal{A}$ has no advantage in triggering the event $abort_{win}$ due to π_i^s processing c_2, c_3.

$$\mathsf{Adv}^{\mathsf{fACCE},\ \textbf{Case B}}_{\mathtt{XK}, n_P, n_S, \mathcal{A}} \leq \mathsf{Adv}^{\mathsf{coll}}_{\text{H}, \mathcal{B}_{\mathsf{coll}}} + n_P^2 n_S^2 \cdot \left(\mathsf{Adv}^{\mathsf{sym\text{-}ms\text{-}PRF\text{-}ODH}}_{\text{KDF}, G, p, \mathcal{B}_{\mathsf{PRF\text{-}ODH}}} + \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{AEAD}, \mathcal{B}_{\mathsf{aead}}} \right)$$

We can now treat **Case C**.

We follow now-standard procedure and define an abort event to trigger when we find a hash-collision, guess the index (i, s) of the session π_i^s, and the index (j, t) of the honest partner π_j^t. By **Case A** and **Case B**, there *must* exist such an honest partner for the beginning of stage $\varsigma = 3$. In what follows, we assume without loss of generality that π_i^s is the initiator session. The analysis where π_i^s is the responder session follows identically, except for a change in notation.

At this point, we need to split the analysis into two sub-cases:

1. **Case C.1**: $\mathcal{A}$ has not issued a $\mathsf{OCorrupt}(j)$ query during the experiment. This allows us to prove the security of all stages ciphertexts.
2. **Case C.2**: $\mathcal{A}$ has issued a $\mathsf{OCorrupt}(j)$ query *after* π_i^s decrypts g^b, c_1 successfully (outputting $\varsigma = 2$). Note that if $\mathcal{A}$ issues a $\mathsf{OCorrupt}(j)$, then $\pi_i^s.fr_1 \leftarrow 0$, and thus $\mathcal{A}$ has no advantage in outputting $(i, s, 1, b')$. This allows us to prove the security of ciphertexts belonging to stages $\varsigma \geq 2$. Note that if $\mathcal{A}$ did not ever issue a $\mathsf{OCorrupt}(j)$ query, then the security analysis reverts to **Case C.1** since $\pi_i^s.fr_1 = 1$, and we need to capture the security of the additional stages' ciphertext.

In **Case C.1 Game 4**, we replace ck, k_0 by uniformly random $\widetilde{ck}, \widetilde{k_0}$ in π_i^s and its honest partner which is reduced to the ms-PRF-ODH assumption (the challenger here acts as in **Case A, Game 5**). Here the session state is again independent of other non-partnered sessions' states. In **Game 5** and **Game 6**, we replace the values $ck, k_1 \leftarrow \mathrm{KDF}(\widetilde{ck}, g^{ab}, 2)$ with uniformly random values $\widetilde{ck}, \widetilde{k_1}$, and subsequently replace $ck, k_2 \leftarrow \mathrm{KDF}(\widetilde{ck}, g^{Ab}, 2)$ with uniformly random values $\widetilde{ck}, \widetilde{k_2}$ via the prf assumption on KDF. Similarly, in **Game 7** we replace $k_{\mathtt{i}}, k_{\mathtt{r}} \leftarrow \mathrm{KDF}(\widetilde{ck}, \epsilon, 2)$ with uniformly random values $\widetilde{k_{\mathtt{i}}}, \widetilde{k_{\mathtt{r}}}$.

In **Case C.1 Game 8** the challenger flips a bit $\bar{b}$ and uses $\bar{b}$ instead of $\pi_i^s.b_1$ when responding to $\mathsf{OEnc}(i, s, m_0, m_1)$ queries from $\mathcal{A}$ when Enc and Dec would output $\varsigma = 1$ (i.e. when using the key $\widetilde{k_0}$ replaced in **Game 4**). Specifically, the challenger constructs an algorithm $\mathcal{B}_{\mathsf{aead}}$ that interacts with an AEAD challenger in the following way: $\mathcal{B}_{\mathsf{aead}}$ acts exactly as in **Game 7** except responding to $\mathsf{OEnc}(i, s, m_0, m_1)$ or $\mathsf{ODec}(j, t, c)$ queries directed to π_i^s (or π_j^t respectively) when π_i^s or π_j^t would output $\varsigma = 1$ and instead forwards the queries to the AEAD challenger's oracles. Since $\widetilde{k_0}$ is a uniformly random and independent value (by **Game 4**), this change is sound.

Case C.1 Game 9 and **Game 10** proceed identically to **Game 8** but flip and use independent challenge bits when answering queries to OEnc if key $\widetilde{k_1}$ is used in stage $\varsigma = 2$ (**Game 9**) and when key $\widetilde{k_2}$ is used in stage $\varsigma = 3$ (**Game 10**). These changes in $\mathcal{A}$'s advantage are bound by the advantage in breaking the underlying aead assumption. Finally, in **Game 11** keys $\widetilde{k_{\mathtt{i}}}, \widetilde{k_{\mathtt{r}}}$ (replaced in **Game 7**) are used in stage $\varsigma = 4$. These changes in $\mathcal{A}$'s advantage are bound by the advantage in breaking the underlying aead assumption. In **Case C.1**, **Game 11**, the behaviour of π_i^s is independent on the test bits $\pi_i^s.b_\varsigma$ (where $\varsigma \geq 1$) and thus $\mathcal{A}$ has no advantage in guessing these challenge bits nor in causing π_i^s to set $\mathsf{win} \leftarrow 1$.

We now treat **Case C.2**, where $\mathcal{A}$ potentially has issued a $\mathsf{OCorrupt}(j)$ query. Since $\mathtt{fs} = 2$, by Table 1 any adversary that issues a $\mathsf{OCorrupt}(j)$ sets $\pi_i^s.fr_1 \leftarrow 0$ and outputting $(i, s, 1, b')$ will lose $\mathcal{A}$ the game. Thus in **Case C.2** we do not prove the security of the first ciphertext's payload data.

Case C.2 Game 4 requires additional care: Note that by **Game 2**, we know at the beginning of the experiment the index of session π_i^s such that (i, s, ς', b') is output by the adversary and by **Game 3**, we know at the beginning of the experiment the index of the honest partner session (j, t) of the session π_i^s. We take

a similar approach to **Game 5** of **Case A**. However, in this game we replace the computation of $ck, k_1 \leftarrow \mathrm{KDF}(ck, g^{ab}, 2)$ with uniformly random and independent values $(\widetilde{ck}, \widetilde{k_1})$ in the test session and its honest partner. Specifically, we define an algorithm $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ that initialises a $\mathsf{sn\text{-}PRF\text{-}ODH}$ challenger, embeds the DH challenge keyshares g^u into the ephemeral public-key of party i (g^a), embeds the DH challenge keyshare g^v into the ephemeral public-key of session π_j^t (g^b), and replaces the computation of $ck, k_1 \leftarrow \mathrm{KDF}(ck, g^{ab}, 2)$ (in the session π_i^s and its partner) with uniformly random values $\widetilde{ck}, \widetilde{k_1}$. Note that $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ can use its internal knowledge of the long-term private keys of party i and party j to compute $(ck, k_0) \leftarrow \mathrm{KDF}(ck, g^{uB}, 2)$ and $(ck, k_2) \leftarrow \mathrm{KDF}(ck, g^{Av}, 2)$. However, $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ must account for $\mathcal{A}$ to issue $\mathsf{OCorrupt}(j)$ *after* π_j^t has computed the ciphertext (g^b, c_1) and instead delivering $(g^{b'}, c_1')$ to π_i^s. To simulate this computation, $\mathcal{B}_{\mathsf{PRF\text{-}ODH}}$ must instead use the ODHu oracle provided by the $\mathsf{sn\text{-}PRF\text{-}ODH}$ challenger, specifically querying $\mathsf{ODHu}(ck, X)$, (where X is the Diffie-Hellman public keyshare such that the private key is unknown to the challenger) which will output $\mathrm{KDF}(ck, X^u)$. Thus we bound the probability of $\mathcal{A}$ distinguishing this change by the security of the $\mathsf{sn\text{-}PRF\text{-}ODH}$ assumption. Due to this game hop π_i^s's session state is independent of other non-partnered sessions' states.

In **Case C.2 Game 5** and **Game 6**, we replace $ck, k_2 \leftarrow \mathrm{KDF}(\widetilde{ck}, g^{Ab}, 2)$ with uniformly random values $\widetilde{ck}, \widetilde{k_2}$, and subsequently replace $k_{\mathtt{i}}, k_{\mathtt{r}} \leftarrow \mathrm{KDF}(\widetilde{ck}, \epsilon, 2)$ with uniformly random values $\widetilde{k_{\mathtt{i}}}, \widetilde{k_{\mathtt{r}}}$. **Case C.2**, **Game 7** proceeds similarly to **Case C.1**, **Game 9** by encrypting $m_{\bar{b}}$ for a randomly flipped bit $\bar{b}$ when Enc and Dec would output $\varsigma = 2$ (i.e. when using the key $\widetilde{k_1}$). **Case C.1**, **Game 8** proceeds similarly to **Case C.1**, **Game 10** by encrypting $m_{\bar{b}}$ for a randomly flipped bit $\bar{b}$ when Enc and Dec would output $\varsigma = 3$ (i.e. when using the key $\widetilde{k_2}$). Finally, **Game 9** proceeds identically to **Case C.1 Game 11** by encrypting $m_{\bar{b}'}$ for another randomly flipped bit $\bar{b}'$ when Enc and Dec would output $\varsigma = 4$ (i.e. when using the keys $\widetilde{k_{\mathtt{i}}}, \widetilde{k_{\mathtt{r}}}$). In **Case C.2**, **Game 9**, the behaviour of π_i^s is independent of the test bits $\pi_i^s.b_\varsigma$ (where $\varsigma \geq 2$) and thus $\mathcal{A}$ has no advantage in guessing these challenge bits nor in causing π_i^s to set $\mathsf{win} \leftarrow 1$. Thus:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{fACCE},\mathbf{Case\ C}}_{\mathtt{XK},n_P,n_S,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{coll}}_{\mathrm{H},\mathcal{B}_{\mathsf{coll}}} + n_P^2 n_S^2 \cdot \Big(\max\Big\{ &\big(3 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathrm{KDF},\mathcal{B}_{\mathsf{prf}}} + \mathsf{Adv}^{\mathsf{ms\text{-}PRF\text{-}ODH}}_{\mathrm{KDF},G,p,\mathcal{B}_{\mathsf{PRF\text{-}ODH}}} \\ &+ 4 \cdot \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{AEAD},\mathcal{B}_{\mathsf{aead}}}\big), \\ &\big(2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathrm{KDF},\mathcal{B}_{\mathsf{prf}}} + 3 \cdot \mathsf{Adv}^{\mathsf{aead}}_{\mathsf{AEAD},\mathcal{B}_{\mathsf{aead}}} + \mathsf{Adv}^{\mathsf{sn\text{-}PRF\text{-}ODH}}_{\mathrm{KDF},G,p,\mathcal{B}_{\mathsf{PRF\text{-}ODH}}}\big)\Big\}\Big).\end{aligned}$$

7 Discussion

The aim of our model is explicitly not to propose the next *super-strong* notion of security (since all security properties can be analyzed optionally but not all independently), but to propose a generic model- and proof-approach.

As the main reason for basing a protocol analysis on an ACCE model is the intertwined design of the specific analyzed protocol (i.e., an atomic channel

establishment), it is surprising that all previous ACCE model definitions were heavily influenced by the concept of composing a channel establishment protocol cleanly from key exchange and channel. Consequently, our results systematize and contribute to the understanding of the generic, composition-independent primitive *authenticated and confidential channel establishment*.

Acknowledgments We thank Trevor Perrin, Sebastian Lauer, Sven Schäge, Bertram Poettering, Marc Fischlin, members of the SKECH workshop 2018, and the reviewers for insightful comments and discussions.

Bibliography

[1] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. DROWN: breaking TLS using sslv2. In *USENIX Security*, 2016.

[2] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO*, 1993.

[3] Florian Bergsma, Benjamin Dowling, Florian Kohlar, Jörg Schwenk, and Douglas Stebila. Multi-ciphersuite security of the secure shell (SSH) protocol. In *CCS*, 2014.

[4] Olivier Blazy, Angèle Bossuat, Xavier Bultel, Pierre-Alain Fouque, Cristina Onete, and Elena Pagnin. Said: Reshaping signal into an identity-based asynchronous messaging protocol with authenticated ratcheting. In *IEEE EuroS&P*, 2019.

[5] Jacqueline Brendel, Marc Fischlin, Felix Günther, and Christian Janson. PRF-ODH: relations, instantiations, and impossibility results. In *CRYPTO*, 2017.

[6] Chris Brzuska and Håkon Jacobsen. A modular security analysis of EAP and IEEE 802.11. In *PKC*, 2017.

[7] Chris Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of bellare-rogaway key exchange protocols. In *CCS*, 2011.

[8] Christina Brzuska, Nigel P. Smart, Bogdan Warinschi, and Gaven J. Watson. An analysis of the EMV channel establishment protocol. In *CCS*, 2013.

[9] Christina Brzuska, Håkon Jacobsen, and Douglas Stebila. Safely exporting keys from secure channels - on the security of EAP-TLS and TLS key exporters. In *EUROCRYPT*, 2016.

[10] Shan Chen, Samuel Jero, Matthew Jagielski, Alexandra Boldyreva, and Cristina Nita-Rotaru. Secure communication channel establishment: Tls 1.3 (over tcp fast open) vs. quic. Cryptology ePrint Archive, Report 2019/433, 2019. `https://eprint.iacr.org/2019/433`.

[11] Katriel Cohn-Gordon, Cas J. F. Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *IEEE EuroS&P*, 2017.

[12] Jason A. Donenfeld. Wireguard: Next generation kernel network tunnel. In *NDSS*, 2017.

[13] Benjamin Dowling and Kenneth G. Paterson. A cryptographic analysis of the wireguard protocol. In *ACNS*, 2017.

[14] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In *CCS*, 2015.

[15] Benjamin Dowling, Paul Rösler, and Jörg Schwenk. Flexible authenticated and confidential channel establishment (facce): Analyzing the noise protocol framework. Cryptology ePrint Archive, Report 2019/436, 2019. https://eprint.iacr.org/2019/436.

[16] Marc Fischlin and Felix Günther. Multi-stage key exchange and the case of google's QUIC protocol. In *CCS*, 2014.

[17] Marc Fischlin and Felix Günther. Replay attacks on zero round-trip time: The case of the TLS 1.3 handshake candidates. In *IEEE EuroS&P*, 2017.

[18] Florian Giesen, Florian Kohlar, and Douglas Stebila. On the security of TLS renegotiation. In *CCS*, 2013.

[19] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-rtt key exchange with full forward secrecy. In *EUROCRYPT*, 2017.

[20] Britta Hale, Tibor Jager, Sebastian Lauer, and Jörg Schwenk. Simple security definitions for and constructions of 0-rtt key exchange. In *ACNS*, 2017.

[21] WhatsApp Inc. Whatsapp encryption overview, 2016. URL https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf. White paper.

[22] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of tls-dhe in the standard model. In *CRYPTO*, 2012.

[23] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Authenticated confidential channel establishment and the security of TLS-DHE. *J. Cryptology*, 30(4), 2017.

[24] Nadim Kobeissi. Noise explorer, 2018. URL https://noiseexplorer.com/.

[25] Nadim Kobeissi, Georgio Nicolas, and Karthikeyan Bhargavan. Noise explorer: Fully automated modeling and verification for arbitrary noise protocols. In *IEEE EuroS&P*, 2019.

[26] Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of tls-dh and tls-rsa in the standard model. Cryptology ePrint Archive, 2013. https://eprint.iacr.org/2013/367.

[27] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In *CRYPTO*, 2013.

[28] Benjamin Lipp, Bruno Blanchet, and Karthikeyan Bhargavan. A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol. In *IEEE EuroS&P*, 2019.

[29] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How secure and quick is quic? provable security and performance analyses. In *IEEE S&P*, 2015.

[30] Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. A modular security analysis of the TLS handshake protocol. In *ASIACRYPT*, 2008.

[31] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This poodle bites: Exploiting the ssl 3.0 fallback, 2014. URL https://www.openssl.org/~bodo/ssl-poodle.pdf.

[32] Trevor Perrin. The noise protocol framework, 2017. URL http://noiseprotocol.org/noise.html. Revision 33.

[33] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: On the end-to-end security of group chats in signal, whatsapp, and threema. In *IEEE EuroS&P*, 2018.