Rethinking Zero-Knowledge TLS: Proof of Protocol Execution and the Reclaim Protocol

Andrey Bozhko[†] and Kirill Kutsenok[⋆]

[†]University of Luxembourg
andrey.bozhko@uni.lu
[⋆]Reclaim Protocol and King's College London
kirill@reclaimprotocol.org

November 19, 2026

Abstract

In a zkTLS protocol, a user proves to a third party a property of data obtained from an online service over TLS, revealing nothing beyond the statement proven. Each existing scheme, however, comes with its own security model, written for that scheme alone, so the schemes are not proven secure against a common definition. The root cause is a mismatch of levels: existing models are stated at the level of TLS and therefore reflect how each construction handles it. Yet TLS is only the transport, and a zkTLS proof is about the application-layer protocol executed over it. We therefore introduce the first transport-agnostic security model for zkTLS, stated at the application layer and mentioning neither TLS nor any particular construction. The resulting notion, *Zero-Knowledge Proof of Protocol Execution* (zkPoPE), defines what it means for a party that has executed an arbitrary two-party protocol with a counterparty to prove, in zero knowledge, that its private input and the reply it received satisfy a chosen property, and, crucially, that the reply comes from a real execution with that counterparty rather than being fabricated. This proof-of-execution requirement separates zkPoPE from a NIZK over the input–reply pair: knowing a satisfying pair does not by itself show that an execution occurred. We formalize zkPoPE as a UC ideal functionality $\mathcal{F}_{\mathsf{zkPoPE}}$. We then introduce the *Reclaim Protocol*, a zkTLS scheme that UC-realizes $\mathcal{F}_{\mathsf{zkPoPE}}$ under a deployment-aligned trust model and underlies a production system. Reclaim builds on *Distributed-AEAD*, a new three-party authenticated-encryption primitive of independent interest. On a lower-mid-range phone, Reclaim attests to a 1200 kB response in 7.28 s, two orders of magnitude beyond any size previously benchmarked for zkTLS.

Contents

1 Introduction **3**

2 Technical Overview **5**
2.1 Zero-Knowledge Proof of Protocol Execution . . . 5
2.2 Reclaim Protocol . . . 7
2.3 Open Problems . . . 11

3 Zero-Knowledge Proof of Protocol Execution **11**
3.1 System Model and Parties . . . 11
3.2 Ideal Functionality . . . 12
3.3 Relation to Existing Models . . . 14

4 Distributed AEAD **14**
4.1 Syntax . . . 15
4.2 Security . . . 16

5 The Reclaim Protocol **16**
5.1 Hybrid model and interface . . . 16
5.2 Admissible Protocols and Predicates . . . 16
5.3 Protocol $\mathcal{RP}$ . . . 18
5.4 Security . . . 20

6 Benchmarks **21**

A Message Flow of the Reclaim Protocol **26**

B Preliminaries **26**
B.1 Notation . . . 26
B.2 Request–Reply Protocols and Request Extractability . . . 27
B.3 Ideal Functionalities . . . 29

C Security of Distributed-AEAD **31**
C.1 Security Properties . . . 32
C.2 Assumptions on the underlying primitives . . . 34
C.3 Main theorem . . . 35
C.4 Concrete schemes . . . 37

D Reclaim Protocol Security Proof **38**
D.1 Model and conventions . . . 38
D.2 Simulator . . . 39
D.3 Indistinguishability: Case B (soundness) . . . 41
D.4 Indistinguishability: Cases A, C, D (zero-knowledge) . . . 42
D.5 Wrap-up . . . 43

1 Introduction

Zero-knowledge TLS (zkTLS) protocols, also known as TLS oracles [ZMM+20], allow a User acting as the TLS client to convince a third party of statements about the data exchanged over a TLS connection. A zkTLS proof attests that the session was established with a specific server, identified by its TLS certificate, and that the application data exchanged over it satisfies a public predicate Φ, while revealing nothing further about the connection or its transcript. In the canonical example, a proof of solvency, a User logs into their bank's web portal over TLS and proves that their balance exceeds a threshold T, revealing neither their credentials, nor their account number, nor the exact balance. Any verifier, be it a lender, an exchange, or a KYC provider, can accept this proof without involving the User or the bank in the verification. The zero-knowledge component is not optional: TLS-protected traffic typically carries credentials, session cookies, and unrelated personal data alongside the value being proved, so every realistic use of zkTLS must hide nearly the entire session.

The TLS transcript is not itself such a proof. TLS [RD08,Res18] authenticates the server to the client and protects application data against a network adversary, but its authentication binds the two endpoints alone. After the handshake, the User holds the symmetric keys for *both* directions and can produce a syntactically valid TLS record stream that decrypts to any plaintext of its choice. A transcript captured by the User is therefore not, by itself, evidence of what the server sent. One way to close this gap is to have the server sign its responses, and a standard for it exists in HTTP Message Signatures [BRS24]. Mechanisms in this spirit are deployed: Stripe authenticates the webhook events it delivers [Str], but what such mechanisms call a signature is a MAC under a secret shared between the two endpoints, and it convinces the counterparty alone, since anyone able to verify it could equally have produced it. Third-party-verifiable response signing has seen no broad deployment, and none as a general mechanism for arbitrary websites. The obstacle is usually the provider, which has no incentive to make the data it serves verifiable by third parties, and equally none to run any other modified software for the benefit of Users or third parties. zkTLS designs therefore impose *server-obliviousness*: to the server, the execution must look like an ordinary TLS session.

With the server unavailable and the client able to forge the transcript on its own, evidence must come from elsewhere: every known zkTLS protocol introduces auxiliary parties, which we call *attestors*, to ensure that the User cannot emit a proof inconsistent with what was exchanged. The *Reclaim Protocol*, which we introduce and analyze, has two attestors, each running in a trusted execution environment (TEE).

A security analysis of Reclaim needs a security definition, and none in the literature serves. Since DECO [ZMM+20], zkTLS constructions come with their own ideal functionalities or bespoke models, each written for one construction and bound to the specifics of TLS, a habit we call *protocol-inseparability*. DECO's definition, for instance, requires the server side of any realizing protocol to run standard TLS, and ORIGO's security theorem [ELW+24] is stated for a single TLS 1.3 cipher suite. Two zkTLS protocols are thus hard to compare, since they realize distinct functionalities, and a construction not built over TLS cannot be evaluated against this literature at all.

The root of this habit is a mismatch of levels. TLS is, in the end, only the transport layer over which a User and a server execute some application-layer protocol[1], such as a bank's balance-inquiry flow. The property a User wishes to prove about a TLS session is a property of *that* application-layer execution rather than of the TLS bytes carrying it. Stated at the level of the application-layer protocol, the definition does not mention the transport, and the same problem statement covers other secure transports (e.g., IPsec [FK11] or QUIC [IT21]), exchanges that no transport signs for third parties, such as a conversation with an AI service, and even deniable channels such as end-to-end encrypted messaging [CGCD+17]. The latter are not a counterexample but the extreme case: the transcript proves nothing by design, so all evidentiary weight falls on the proof-of-execution machinery. The common object is a party's participation in a protocol and

[1] We use *application-layer* for everything above the transport: the message syntax carrying the exchange, such as HTTP, *together with* the business logic offered through it. The two are conceptually separable, and the IETF's guidance treats the latter, which has no settled name of its own, as an application built on HTTP as a "substrate" [Not22]. We take them as a single object, since the messages of such a protocol are by definition the byte strings handed to the transport, and the properties we prove of them are stated over those bytes.

the wish to prove a property of it in zero knowledge, whatever the transport. To our knowledge, no zkTLS security definition has been stated at this level.

Stating the definition at this level is the central move of this paper. We call the resulting notion *Zero-Knowledge Proof of Protocol Execution* (zkPoPE), and the conventional zkTLS problem is then its instance in which the proven protocol runs over TLS.[2] zkPoPE is closely related to the problem NIZKs solve, and Section 2 obtains its functionality as a minimal adaptation of the NIZK one. It is nonetheless not a NIZK: its language cannot be checked without the counterparty's private state, it admits no meaningful witness extraction, and a proof must attest to an *interaction* rather than to a computation the prover could run alone. Each difference forces a departure from the NIZK template. Beyond that, the definition restores comparability, since constructions for all the settings above become candidate realizations of a single notion, and it is forward-compatible: whatever transport eventually displaces TLS, and however deeply the post-quantum migration reshapes the cryptography beneath it [Con26], the model stays as it is.

Reclaim's trust assumptions expose a second gap in the zkTLS literature. Existing constructions form two families. *Cryptographic-attestor* designs use no trusted hardware: *proxy* variants observe TLS ciphertexts on the wire and commit to them before the User releases the key material [LEFS23, ELW+24], while *MPC* variants compute the handshake and record encryption jointly with the User, so that neither party alone holds the traffic keys [ZMM+20, KK24, XYWY23, CDH+25]. Here theorems and deployments describe different situations: the theorems are stated for n attestors under a k-of-n honesty assumption, while the protocols are deployed with two or three of them. Closing this gap requires an argument that the security analysis does not supply, namely that the named attestors are independently operated, have an economic stake in honest behavior, and so on. The substantive trust assumption is thus the behavioral honesty of a specific small committee, justified outside the protocol. The *TEE-based* family, exemplified by Town Crier [ZCC+16], runs the TLS client inside a TEE such as Intel SGX [Int15] and emits signed transcripts whose authenticity reduces, via remote attestation, to the integrity of the TEE. It is proven and deployed in the same configuration, but relies on the integrity *and confidentiality* of an entire TEE-resident TLS client. Confidentiality is required since the transported data is itself sensitive, the only case in which zkTLS is of interest at all. The long line of side-channel and microarchitectural attacks on commodity TEEs [VBMW+18, vSKGY20, CSB+26] has made this confidentiality assumption hard to maintain, and the approach has been left dormant. Each family nonetheless has an element worth preserving: cryptographic-attestor designs spread trust across several parties, so that no single party holds material catastrophic if compromised, while the TEE-based approach is proven and deployed in the same configuration. What is missing is a construction that keeps both elements without the problematic assumptions just discussed and without the cost the cryptographic-attestor branch imposes on the User, whose device runs multi-party computation or zero-knowledge proof generation at a cost that grows sharply with the amount of data proven.

This work makes two contributions: the security model and a zkTLS construction that realizes it in the TLS setting. More specifically:

(1) The first transport-agnostic security model for zkTLS: zkPoPE. The model is stated for two-party protocol executions, independently of construction, transport, and application-layer protocol. We define, in the UC framework [Can01], an ideal functionality $\mathcal{F}_{\mathsf{zkPoPE}}$ for the User, Service, and Verifier roles, obtained from the NIZK functionality through the departures that proof of execution forces. We regard the introduction of zkPoPE and of $\mathcal{F}_{\mathsf{zkPoPE}}$ as the principal conceptual contribution of this work, because the object it isolates, proof of execution, is one that no proof system for a locally checkable relation can express.

(2) The Reclaim Protocol. We present the *Reclaim Protocol*,[3] a zkTLS protocol that builds on a new three-party authenticated-encryption primitive, *Distributed-AEAD*. The latter splits the encryption of a record under unmodified AES-GCM or ChaCha20-Poly1305 among three roles, so that the key holder never sees the bytes the data owner marks secret, while the data owner never holds the key. We prove it secure

[2] Strictly applied, the naming logic of zkPoPE would have zkTLS denote a scheme proving execution of TLS itself, whereas existing zkTLS schemes, Reclaim included, prove execution of an application-layer protocol that runs over TLS. We retain zkTLS in line with established usage.

[3] An informal description of an early version of the Reclaim Protocol appeared in a whitepaper [Pro25] on the deploying company's website. It contains none of the security model, Distributed-AEAD, functionality, or proof presented here.

under standard assumptions on those primitives. Reclaim runs against an unmodified TLS server, and we prove it UC-realizes $\mathcal{F}_{\mathsf{zkPoPE}}$ under stated assumptions on its two attestors and for an explicitly limited class of protocols and predicates: request–reply protocols from which the User's input can be extracted, as holds for public request formats such as HTTP, with predicates over the request and reply bytes the User has chosen not to hide. It keeps both elements identified above: trust spread across two attestors, and proof and deployment in the same configuration under narrowly scoped TEE guarantees. The User in the Reclaim Protocol runs neither multi-party computation nor proof generation, and its cost scales only with the size of the response rather than with the amount of data proven. On a lower-mid-range phone, proving responses of up to 25 kB takes about 3 s end to end, and a 1200 kB response 7.28 s, against a previous zkTLS maximum of 8 kB in 6.2 s [LEFS23]. Reclaim underlies a deployed production system, but we claim no software contribution.

Organization. Section 2 gives a technical overview of the contributions. Section 3 formalizes zkPoPE and $\mathcal{F}_{\mathsf{zkPoPE}}$ and relates them to existing models (Section 3.3). Section 4 introduces Distributed-AEAD, Section 5 specifies Reclaim, and Section 5.4 states the theorem with a proof sketch. Section 6 reports the benchmarks. Appendices hold the preliminaries (Appendix B), the Distributed-AEAD security analysis (Appendix C), and the full proof (Appendix D).

2 Technical Overview

This section gives an informal account of the two contributions. Section 2.1 derives the zkPoPE definition: we fix the setting, state what the User claims, identify the two requirements this claim places on a proof system, and obtain the ideal functionality $\mathcal{F}_{\mathsf{zkPoPE}}$ from the NIZK one through the departures those requirements force. Section 2.2 describes Reclaim, our realization of $\mathcal{F}_{\mathsf{zkPoPE}}$ over TLS, and the Distributed-AEAD primitive at its core.

2.1 Zero-Knowledge Proof of Protocol Execution

2.1.1 Setting

As argued in Section 1, the object of a zkTLS proof is the execution of an application-layer protocol, and the transport carrying it does not enter the statement. We model the participants at that level and restrict to two-party protocols, which covers the zkTLS class, and we call the parties U and $\mathsf{Service}$.[4]

Let $\mathcal{US} = \{\mathsf{Service}_{\mathsf{id}}\}_{\mathsf{id}\in\mathcal{ID}}$ be a universe of Services. Each $\mathsf{Service}_{\mathsf{id}}$ is initialized by some setup procedure that produces a triplet $(\mathsf{id}, \mathsf{sec}_{\mathsf{id}}, \mathsf{Prot}_{\mathsf{id}})$: a unique identifier, the Service's private state (internal databases, per-user records), and a public description of the protocol a User can run with it. We write $\mathsf{Service}_{\mathsf{id}}$ for both the Service and its triplet. Besides the Services, there are a User U and a Verifier V.

The User runs $\mathsf{Prot}_{\mathsf{id}}$ with $\mathsf{Service}_{\mathsf{id}}$ on a private input data of the format $\mathsf{Prot}_{\mathsf{id}}$ expects[5] and obtains

$$\mathsf{reply} = \mathsf{out}_{\mathsf{U}}\big(\mathsf{Prot}_{\mathsf{id}}\langle \mathsf{Service}_{\mathsf{id}}(\mathsf{sec}_{\mathsf{id}}),\ \mathsf{U}(\mathsf{data})\rangle\big),$$

its local output at the end of the interaction. The Service obtains a local output $\mathsf{out}_{\mathsf{Service}}$, which may update $\mathsf{sec}_{\mathsf{id}}$. The User then wishes to prove a property of $(\mathsf{data}, \mathsf{reply})$ to the Verifier. The two agree, possibly out of band, on a predicate Φ over the input–reply space of $\mathsf{Prot}_{\mathsf{id}}$.[6] Φ may depend on parts of data the User does not wish to hide, such as the public part of a query, while credentials stay secret. The User's goal is to convince the Verifier that the protocol was executed and that $\Phi(\mathsf{data}, \mathsf{reply}) = 1$.

[4] We write "Service" rather than "server" since the object of interest is the application-layer functionality, not the host running it. Protocols $\mathsf{Prot}_{\mathsf{id}}$ with more than two parties are left to future work (Section 2.3).

[5] We make no assumption about how the input is chosen. Inputs of incorrect format are rejected by the Service as per the specification of $\mathsf{Prot}_{\mathsf{id}}$.

[6] Φ may be specific to one User–Verifier pair or standardized across many sessions of $\mathsf{Prot}_{\mathsf{id}}$, for example, a predicate fixed for a class of governmental web services with a common interface. We make no assumption either way.

2.1.2 Two requirements

In proof-system terms, the User proves knowledge of an input data for which Φ holds on the reply of an execution of $\mathsf{Prot}_{\mathsf{id}}$ with $\mathsf{Service}_{\mathsf{id}}$. The statement is (id, Φ), and we write $\mathcal{L}_{\mathcal{US}}$ for the language of such statements (Section 3.1). The witness is data alone: the reply is determined by the execution, not supplied by the prover. The relation cannot be evaluated without $\mathsf{sec}_{\mathsf{id}}$, so membership is relative to the deployment $\mathcal{US}$ rather than to the public descriptions.

The first requirement is zero-knowledge in the standard sense: the proof reveals nothing about $(\mathsf{data}, \mathsf{reply})$ beyond Φ and the fact that the pair satisfies it. This fixes what "knowledge" means in the "zk" of zkTLS and zkPoPE.

The second requirement is that the proof attest to an execution of $\mathsf{Prot}_{\mathsf{id}}$ between the User and $\mathsf{Service}_{\mathsf{id}}$, for an *arbitrary* application-layer protocol. No proof system can check the relation of $\mathcal{L}_{\mathcal{US}}$ directly, since that needs $\mathsf{sec}_{\mathsf{id}}$. The fallback, a NIZK for the locally checkable relation "$\Phi(\mathsf{data}, \mathsf{reply}) = 1$" on a User-supplied pair, certifies knowledge of the pair but not execution. The two coincide for some pairs: if reply carries a Service signature over part of data (as in [BRS24]) and Φ verifies it, knowledge of $(\mathsf{data}, \mathsf{reply})$ is evidence of execution. For most protocols of interest they do not: a reply may be no more than `200 OK`, and a User can satisfy "data contains the recipient's address and reply is `OK`" by writing the pair down, whether or not a transfer took place. This separates zkPoPE from every proof system for a locally checkable relation, and is why every zkTLS scheme adds an execution-binding mechanism (attestors, MPC, or TEEs). We leave the characterization of the pairs for which a NIZK suffices to future work (Section 2.3).

2.1.3 The ideal functionality

zkPoPE thus asks for the zero-knowledge of a NIZK over $(\mathsf{data}, \mathsf{reply})$ together with a guarantee that the pair was produced by an execution of $\mathsf{Prot}_{\mathsf{id}}$ with the registered $\mathsf{Service}_{\mathsf{id}}$. We adapt the NIZK formalization to the second requirement in the UC framework [Can01]. The out-of-band parts of the setting map onto the environment, and the resulting ideal functionality (IF) $\mathcal{F}_{\mathsf{zkPoPE}}$ is a building block for applications consuming zkPoPE proofs.

$\mathcal{F}_{\mathsf{zkPoPE}}$ keeps the interface of the NIZK IF: Prove and Verify with the semantics of [GOS06], and a Setup message as in formulations that place the scheme's algorithms inside the functionality [LR25]. In the NIZK IF:

- Setup initializes the functionality, after which all parties hold a CRS.
- Prove takes a statement and a witness from a party and checks them against the relation. On success, it obtains a proof from the simulator Sim, returns it to the party, and records it. Drawing the proof from Sim captures zero-knowledge.
- Verify takes a statement and a proof and checks them against the record. If they are not recorded, it asks Sim for a witness and checks the relation on it. This captures knowledge soundness.

$\mathcal{F}_{\mathsf{zkPoPE}}$ departs from this template at three points, one for each difference from NIZK named in Section 1. The formal specification is given in Figure 2.

Setup is relative to a deployment. Since membership in $\mathcal{L}_{\mathcal{US}}$ depends on the Services' private states, Setup is invoked twice. A Service first registers its triplet $(\mathsf{id}, \mathsf{sec}_{\mathsf{id}}, \mathsf{Prot}_{\mathsf{id}})$ and is added to the universe $\mathcal{US}_{\mathsf{sid}}$. Only afterwards do the remaining parties invoke Setup, supplying the set of Services they expect to interact with. The IF checks it against $\mathcal{US}_{\mathsf{sid}}$ and asks the simulator for the scheme's algorithms and the CRS.

Prove runs the protocol. This is the central modification and carries both requirements. A User-supplied $(\mathsf{data}, \mathsf{reply})$ does not witness execution, so $\mathcal{F}_{\mathsf{zkPoPE}}$ accepts no reply from the User: on input $(\mathsf{sid}, \mathtt{Prove}, \mathsf{id}, \Phi, \mathsf{data})$ it runs $\mathsf{Prot}_{\mathsf{id}}$ itself between $\mathsf{Service}_{\mathsf{id}}(\mathsf{sec}_{\mathsf{id}})$ and $\mathsf{U}(\mathsf{data})$, hands the genuine reply to the User and $\mathsf{out}_{\mathsf{Service}}$ to the Service, evaluates $\Phi(\mathsf{data}, \mathsf{reply})$, and only on success asks the simulator for a proof. The simulator learns (P, Φ, id), the verdict, and the length of reply, but never data or reply: this is the zero-knowledge content. The reply comes from an interaction with the registered $\mathsf{Service}_{\mathsf{id}}$, whose ideal-world view thus matches a real

run: this is the proof-of-execution content. A corrupted prover may name Φ only after receiving reply. Since it holds the pair and could evaluate any predicate on it anyway, nothing false becomes provable.

Verify is record-based. $\mathcal{F}_{\mathsf{zkPoPE}}$ accepts $(P, \Phi, \mathsf{id}, \pi)$ if and only if this tuple appears in the list Π_{sid} of proofs issued via Prove. Soundness holds by construction: every accepted proof was produced from a real execution in which $\Phi(\mathsf{data}, \mathsf{reply}) = 1$. There is no extractor: the relation of $\mathcal{L}_{\mathcal{US}}$ cannot be checked without $\mathsf{sec}_{\mathsf{id}}$, and an extracted witness would not certify execution. A proof is valid because $\mathcal{F}_{\mathsf{zkPoPE}}$ issued it.

2.2 Reclaim Protocol

Reclaim realizes $\mathcal{F}_{\mathsf{zkPoPE}}$ when the transport is TLS, against a Service S_{id} running an unmodified TLS server. Figure 1 shows the parties and their connections.

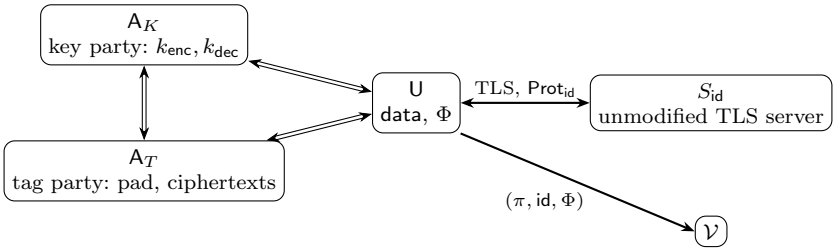

Figure 1: Parties of the Reclaim protocol. U, A_K and A_T are pairwise connected by secure channels $\mathcal{F}_{\mathsf{SC}}$ (double lines); U talks to S_{id} over the public network; $\mathcal{V}$ receives only the proof. Figure 5 in Appendix A gives the message flow.

Supported protocols and predicates. Reclaim supports a restricted class of protocols and predicates: *request–reply* protocols with *extractable* requests. In a request–reply protocol, the User sends one message and the Service answers with one, as in a web-API call. A request is extractable if from it one can recover an input on which the protocol regenerates it. On the predicate side, Reclaim supports conditions on the bytes of the request and the response at positions Φ leaves disclosed, subject to three requirements. The condition must depend on the User's input only through the request that input produces, it must not be satisfied by the reply a Service returns to a malformed request, and it must fix the byte at every position it does not hide, since the Verifier sees all of them.

Extractability is assumed rather than enforced because of Reclaim's cost target: keeping the User's work small rules out having it prove anything about the request it sends. Section 5.2 states the conditions informally alongside the protocol, and Appendix B.2 states them formally.

Distributing trust between two attestors. A single attestor holding the traffic key, decrypting everything, and signing the result would make proof of execution easy and destroy zero-knowledge: it would see data and reply in full. Reclaim splits the job between two attestors, A_K and A_T, and partitions the roles so that no single view suffices to recover the User's session data. Both attestors are semi-honest, and at most one of them is corrupted in any execution, or alternatively the User is malicious and both attestors are honest. A_K holds the traffic key but sees the request only with the private bytes one-time-padded out, while A_T sees the wire ciphertexts and the pad but never the key. The User holds neither the key nor any MAC material, and so cannot forge a record under the session.

Distributed-AEAD. The mechanism behind this separation, and the technical core of Reclaim, is a new primitive, *Distributed-AEAD*. Producing an AES-GCM or ChaCha20-Poly1305 record[7] has two stages. First, a keystream and a MAC subkey are derived from the traffic key and the nonce, independently of the message. Encryption and tagging are then a public computation on these and the plaintext. Schematically, with the

[7]The mandatory-to-implement and recommended cipher suites of TLS 1.3 [Res18] and the dominant choices in deployed TLS 1.2 [RD08].

nonce and associated data left implicit:

$$\begin{array}{ccc} & \text{Distributed-AEAD} & \\ \swarrow & & \searrow \\ \text{encryption key } K & & \text{tag material } \mathsf{TM} \\ \underbrace{C = \mathrm{Enc}(P, K)}_{\substack{\text{encrypt plaintext } P \\ \text{with key } K}} & \xrightarrow{\text{ciphertext } C} & \underbrace{T = \mathrm{MAC}(C, \mathsf{TM})}_{\substack{\text{compute authentication tag } T \\ \text{using tag material } \mathsf{TM}}} \end{array}$$

The key and the plaintext therefore need not co-reside. Distributed-AEAD distinguishes three roles: a *key party*, which holds the traffic key and emits the derived material, a *data party*, which owns the message, and a *tag party*, which combines the two into the ciphertext-and-tag pair.

The data party (the User) one-time-pads the bytes it wants private and sends the padded message to the key party, which thus encrypts a message equal to the original on the public bytes and uniformly random on the private ones. The data party sends the pad to the tag party. Encryption under both cipher suites is an XOR with the keystream, so XOR'ing the pad out of the resulting ciphertext yields the encryption of the *original* message, bit for bit, without exposing the private bytes to the tag party. With K_s denoting the keystream derived from the key and the nonce, the cancellation is explicit:

$$\underbrace{(P \oplus \mathsf{mask}) \oplus K_s}_{\substack{\text{key party encrypts} \\ \text{the masked plaintext}}} \oplus \underbrace{\mathsf{mask}}_{\substack{\text{tag party removes} \\ \text{the mask}}}$$

$$= P \oplus K_s \oplus \underbrace{(\mathsf{mask} \oplus \mathsf{mask})}_{0} = C,$$

$$\mathsf{mask}[i] = \begin{cases} \text{an independent uniform byte}, & i \text{ is private}, \\ 0, & \text{otherwise}. \end{cases}$$

The tag party then tags the recovered ciphertext with the MAC subkey. To the Service, the record looks like one an ordinary TLS client produced.

Decryption of the response runs in reverse, with one ordering constraint: the tag party receives the ciphertext-tag pair, obtains the MAC material from the key party, and verifies the tag, and only then does the key party release the keystream to the User, who decrypts in private. Thus, at the User,

$$\underbrace{\mathit{Resp}}_{\text{response plaintext}} = \underbrace{\mathit{Resp}_{\mathsf{enc}}}_{\substack{\text{verified response} \\ \text{ciphertext}}} \oplus \underbrace{K_{\mathsf{resp}}}_{\substack{\text{keystream released only} \\ \text{after tag verification}}} .$$

When the tag is verified, the User holds no key-derived material, so the verified tag certifies the ciphertext the Service emitted under keys from a server-authenticated handshake. This chain, from certificate through handshake keys to response tag, links the response the User observes to an execution with the registered $\mathsf{Service}_{\mathsf{id}}$, which is what Prove of $\mathcal{F}_{\mathsf{zkPoPE}}$ requires of reply.

The four-phase flow. Reclaim wraps these two Distributed-AEAD calls, one for the request and one for the response,[8] in a four-phase flow. In Phase A, A_K runs the TLS handshake against S_{id} with the User mediating the network. A_K is the cryptographic peer, while the User owns the endpoint S_{id} observes and forwards messages between them. Thus S_{id} sees a vanilla TLS handshake with a vanilla client, and server-obliviousness holds at the level of the program S_{id} executes. Phase B encrypts the request via Distributed-AEAD and forwards the record to S_{id}, and Phase C handles the response by the tag-then-keystream procedure.

The proof transcript. Phase D produces the proof. The redaction sets are those fixed by Φ. The User has named the request set in Phase B and names the response set now. Each attestor signs the slice of

[8] We describe a single request–response exchange: the User obtains session state such as a cookie by ordinary use of the Service, then issues one authenticated request through Reclaim. Multi-round exchanges are a routine extension we omit.

the joint view its role exposes: A_K signs its view of the request, zeroed at the redacted positions, together with the response keystream restricted to the non-redacted positions, and A_T signs the response ciphertext it verified in Phase C, likewise zeroed. Both signatures cover the redaction set and a per-session tag fixed in the handshake, which binds them to one session and one redaction. Neither attestor signs the response plaintext, which neither of them held. Moreover, A_T checks in Phase B that the pad touches only redacted positions, so that A_K's view agrees with the request actually sent everywhere else.

The zeroing matters: A_K's view of the request is the padded plaintext and A_T holds the pad, so a proof carrying the padded bytes at redacted positions would let a corrupted A_T unmask them. The Verifier reads the request-side inputs of Φ off A_K's signed view, which equals the plaintext at the non-redacted positions, and obtains the response-side inputs by XOR'ing A_T's signed ciphertext with A_K's signed keystream there. On the response side, this reconstruction is position-wise:

$$\underbrace{\widehat{Resp}[i]}_{\substack{\text{verifier's reconstructed}\\ \text{response}}} = \underbrace{\widetilde{Resp}_{\mathsf{enc}}[i]}_{\substack{\text{ciphertext signed}\\ \text{by the tag party}}} \oplus \underbrace{\widetilde{K}_{\mathsf{resp}}[i]}_{\substack{\text{keystream signed}\\ \text{by the key party}}}$$

$$= \begin{cases} Resp_{\mathsf{enc}}[i] \oplus K_{\mathsf{resp}}[i] = Resp[i], & i \text{ is disclosed,} \\ 0 \oplus 0 = 0, & i \text{ is hidden.} \end{cases}$$

Φ is then evaluated on the resulting partial views.

Efficiency. Across all parties, Reclaim's computation is that of one TLS execution between A_K and S_{id}, plus the redactions, two signatures, and a few off-path messages. On the User's device, it is a subset of a TLS client's work, namely the application logic and message forwarding, plus sampling a pad, XOR'ing it in, and choosing the redaction set: no multi-party computation and no proof generation, which dominate the User-side cost of every cryptographic-attestor construction, MPC [ZMM+20,KK24,XYWY23] and proxy [LEFS23, ELW+24] alike. The User's cost scales with the size of the response, not with the size of the data proven, so attesting to an entire response costs no more than attesting to one byte of it. On a lower-mid-range phone, Reclaim attests to a 1200 kB response in 7.28 s, whereas the prior zkTLS maximum is 8 kB in 6.7 s [LEFS23]. Full results are in Section 6.

Extension: hiding predicate inputs from the Verifier. The Verifier in Reclaim reconstructs the non-redacted bytes and evaluates Φ on them, so Reclaim provides *selective disclosure*: the Verifier learns the disclosed bytes and the two lengths, and nothing else of the session. Predicates whose inputs must stay hidden, such as thresholds, set membership, or the proof of solvency of Section 1, need a zero-knowledge proof layered between User and Verifier on top of Reclaim. The choice of that layer is orthogonal to our contributions, and we do not pursue it.

We specify Reclaim in Section 5 in a hybrid UC model, where the TLS handshake, certificate verification, and the links among U, A_K, and A_T are ideal functionalities ($\mathcal{F}_{\mathsf{KE}}, \mathcal{F}_{\mathsf{CERT}}, \mathcal{F}_{\mathsf{SC}}$ [CK02, Can04]), and we analyze Distributed-AEAD on its own in Section 4. This separates Reclaim's cryptographic content from TLS message handling, treated elsewhere [GMP+08a, KMO+15].

2.2.1 Security of the Reclaim Protocol

We prove in Section 5.4 that Reclaim UC-realizes $\mathcal{F}_{\mathsf{zkPoPE}}$ in the hybrid model with $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{F}_{\mathsf{KE}}$, $\mathcal{F}_{\mathsf{CERT}}$, and $\mathcal{F}_{\mathsf{US}}$. The first three functionalities abstract the TLS handshake, certificate verification, and the secure links among U, A_K, and A_T, as in the previous subsection. The fourth, $\mathcal{F}_{\mathsf{US}}$, models the U–S_{id} wire, and its behavior depends on whether an attestor is corrupted. With both attestors honest, it is an adversarial network, leaking every wire byte and permitting messages to be modified, reordered, and dropped. With either attestor corrupted, it is a secure channel, leaking only message lengths and delivering every message unaltered. A corrupted A_K holds the session keys, so an adversary that could also read the wire would decrypt the records. A corrupted A_T holds the tag material, so an adversary that could also write to the wire would substitute records of its own, validly tagged.

Corruption pattern. Corruption is static. Writing FC for the fully corrupted parties and HCC for the semi-honestly corrupted ones, a choice (FC, HCC) is *admissible* if

$$FC \subseteq \{\mathsf{U}\}, \quad HCC \subseteq \{\mathsf{A}_K, \mathsf{A}_T\}, \quad |HCC| \leq 1, \quad FC \neq \emptyset \implies HCC = \emptyset.$$

Services are honest, and $\mathcal{V}$ is omitted from FC because proofs are visible to the adversary by default, so a corrupted $\mathcal{V}$ sees nothing new. The bound $|HCC| \leq 1$ is needed because each attestor holds one half of what is required to recover the User's session data, and two corruptions would put both halves in one view. Semi-honesty is needed because a fully malicious A_K holds the traffic key and could manipulate the AEAD keystream, breaking soundness. This is the single-party threat model of Distributed-AEAD (Section 4.2).

Theorem 2.1 (Security of Reclaim; informal statement of Theorem 5.1) Let $\mathsf{D\text{-}AEAD}$ satisfy the confidentiality and ciphertext-integrity guarantees of Section 4.2, let every $\mathsf{Prot}_{\mathsf{id}}$ in the deployment be request–reply and request-extractable, and let every predicate be supported for its protocol. Then $\mathcal{RP}$ UC-realizes $\mathcal{F}_{\mathsf{zkPoPE}}$ in the $(\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{KE}}, \mathcal{F}_{\mathsf{CERT}}, \mathcal{F}_{\mathsf{US}})$-hybrid model under the admissible corruption pattern above.

The statement carries three restrictions, on the protocols, on the predicates, and on the corruptions. Each is forced by a point at which the proof would otherwise fail:

- Request extractability is needed because against a malicious U only a request is observable, and $\mathcal{F}_{\mathsf{zkPoPE}}$ must nonetheless be supplied with an input.
- The conditions on the predicate are needed because $\mathcal{F}_{\mathsf{zkPoPE}}$ evaluates Φ on the User's input while $\mathcal{V}$ evaluates it on (a part of) the transcript, so the two verdicts have to agree, and because $\mathcal{V}$ sees every byte the predicate does not withhold, so those bytes must be fixed by the predicate.
- The corruption pattern is needed for the reason just given.

Beyond the assumptions on Distributed-AEAD and the modeling conventions of Section 5.4, the list is complete, since a UC realization must reproduce $\mathcal{F}_{\mathsf{zkPoPE}}$ in every execution the environment can arrange: whatever the construction needs appears either as a hypothesis or as a step of the proof. A construction-specific security model may leave such conditions implicit.

Only request extractability limits which application-layer protocols a deployment may target. Where the request format is a public serialization, its parser inverts it and the condition holds outright. It must be assumed only for formats that are not publicly invertible, such as those embedding a signature under a U-held key, a commitment, or a zero-knowledge proof. For those, the assumption has content only in an idealized model: in the plain model, the environment may hand the adversary a ready-made request, so the extractor must be an inverter, and the condition collapses to invertibility (Appendix B.2).

Matching the corruption pattern to the deployment. The deployment places each attestor in a TEE under an independent operator. A TEE guarantees *integrity*, that the platform runs the attested code, and *confidentiality*, that enclave state stays hidden. We rely only on the first. Integrity is enforced by remote attestation, whereas attacks extracting enclave secrets [VBMW+18, vSKGY20, CSB+26] are practical, even though each targets specific hardware generations and is either closed by vendor updates or requires physical access to the machine. Integrity ensures that each attestor runs its program, so the only gain against it is reading its state, which is a semi-honest corruption. A breach of one enclave then yields at most one half of Distributed-AEAD's split, the traffic key without the private bytes or the private bytes without the key. The attestors run on distinct architectures under distinct operators, so compromising both takes two attacks, and we treat them as non-colluding. The U is the system's actual user, with an incentive to forge records, so it is fully malicious. Colluding with an attestor would require breaking that attestor's integrity, which is why U is never corrupted alongside one. Town Crier's analysis [ZCC+16] demands the same integrity of an enclave running a full TLS client, and, in addition, confidentiality of its whole view.

From $\mathcal{F}_{\mathsf{CERT}}$ to remote attestation. $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_K]$ and $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_T]$ are instantiated by signing keys that exist only inside the enclaves. At launch, each attestor samples a key pair and places the verification key in a remote-attestation report, which the platform signs over a measurement of the code it runs. A verifier

checks, once per attestor, that the report verifies under the vendor's attestation infrastructure and that its measurement equals the published program, and, per proof, that σ_K and σ_T verify under the keys the reports carry. The signing key is the only enclave state whose secrecy the design requires.

2.3 Open Problems

We list directions naturally suggested by the present work.

Characterizing the zkPoPE landscape. Each construction in this space – Reclaim and the zkTLS protocols surveyed in Section 1 and Section 3.3 – rests on a particular combination of assumptions on the auxiliary parties, on the transport, and on the protocol–predicate pair $(\mathsf{Prot}_{\mathsf{id}}, \Phi)$. A fundamental open direction is to characterize the landscape of zkPoPE solvability as a whole: which choices of $(\mathsf{Prot}_{\mathsf{id}}, \Phi)$ and corruption model admit a zkPoPE realization, and at what minimum communication and computational costs in auxiliary parties. Some attestor-free regimes are partially understood – e.g., protocols whose replies the Service signs, as in RFC 9421 [BRS24] – but the boundary between these regimes and the attestor-bearing regime in which Reclaim and the rest of the existing zkTLS literature operate remains uncharted. Both constructive results that extend the attestor-free regime and impossibility results that delimit it from above would clarify the structure of the problem class.

A universal zkPoPE protocol. The characterization above has a constructive counterpart, which we regard as the central open problem posed by this work: a *universal* zkPoPE protocol, i.e., a single construction serving arbitrary protocol–predicate pairs over arbitrary transports, playing for interactions the role that zkVMs – proof systems for the correct execution of arbitrary programs [BSCG+13] – play for local computation. Reclaim shows that the request–reply TLS instance admits an efficient realization under a deployment-aligned trust model. Whether one construction can serve the whole class, and under which minimal assumptions, remains open.

Beyond two-party protocols. Our formalization of zkPoPE restricts $\mathsf{Prot}_{\mathsf{id}}$ to a two-party interaction between U and $\mathsf{Service}_{\mathsf{id}}$. Extending the model and the surrounding constructions to multi-party protocols, in which any party can prove a property of its participation in a protocol with several other parties, would broaden the class of attestable executions considerably and is a natural complement to the present work.

Increasing adversarial capabilities. The corruption pattern under which Reclaim is proved secure (at most one of A_K, A_T corrupted semi-honestly, or alternatively a malicious U with both attestors honest) was derived by ruling out attacks against stronger adversaries. Reducing the gap between the corruption model under which security can be proved and the natural all-powerful adversary – in particular, accommodating fully malicious attestors – remains open and is likely to require additional cryptographic machinery on top of Distributed-AEAD.

3 Zero-Knowledge Proof of Protocol Execution

3.1 System Model and Parties

We fix the notation for the setting of Section 2.1.

Universe of services: A *universe of services* is a finite set

$$\mathcal{US} \;=\; \big\{\, (\mathsf{id},\, \mathsf{Prot}_{\mathsf{id}},\, \mathsf{sec}_{\mathsf{id}}) : \mathsf{id} \in \mathcal{ID} \,\big\},$$

each entry comprising a public identifier id, a publicly known two-party application-layer protocol $\mathsf{Prot}_{\mathsf{id}}$ that the service runs with a user, and the service's private state $\mathsf{sec}_{\mathsf{id}}$.[9] $\mathcal{US}$ is fixed by an out-of-band setup prior to any zkPoPE session and is treated as a session-level parameter; we write $\mathsf{Service}_{\mathsf{id}}$ for the triple in $\mathcal{US}$ indexed by id.

[9] We keep $\mathsf{sec}_{\mathsf{id}}$ fixed for the lifetime of the deployment for simplicity. A real service's private state might evolve between sessions; membership is then read relative to the state at the session in question, exactly as the service's coins are in Remark 3.1.

Parties: A zkPoPE deployment over $\mathcal{US}$ involves a user U, a verifier V, and the services in $\mathcal{US}$. Concrete realizations may involve auxiliary parties, such as the two attestors of Section 5; these do not appear in the abstract interface.

Predicate and language: A *predicate* Φ is a boolean function over input–reply pairs $(\mathsf{data}, \mathsf{reply})$ of $\mathsf{Prot}_{\mathsf{id}}$, agreed upon out-of-band between U and V. The user's task is to convince the verifier of membership of (id, Φ) in the language

$$\mathcal{L}_{\mathcal{US}} \;=\; \Big\{\, (\mathsf{id}, \Phi) \;\Big|\; \exists\, \mathsf{data} : \Phi\big(\mathsf{data},\, \mathsf{out}_{\mathsf{U}}(\mathsf{Prot}_{\mathsf{id}}\langle \mathsf{Service}_{\mathsf{id}}(\mathsf{sec}_{\mathsf{id}}),\, \mathsf{U}(\mathsf{data})\rangle)\big) = 1 \,\Big\}.$$

Membership in $\mathcal{L}_{\mathcal{US}}$ is defined relative to the specific deployment $\mathcal{US}$, not to the public descriptions $\{(\mathsf{id}, \mathsf{Prot}_{\mathsf{id}})\}_{\mathsf{id}}$ alone, since it depends on the services' private states $\mathsf{sec}_{\mathsf{id}}$. Φ is a public parameter; data is the user's private witness, and comprises exactly what U needs in order to run $\mathsf{Prot}_{\mathsf{id}}$ — credentials, cookies, query parameters — and nothing the protocol does not consume. Membership asks nothing further of Φ; a realization may.[10]

Remark 3.1 We have treated $\mathsf{Prot}_{\mathsf{id}}$ as a deterministic function of $(\mathsf{sec}_{\mathsf{id}}, \mathsf{data})$. The protocols of interest are randomized, and the two sides' coins enter the language differently. The user's tape R_{U} joins data in the witness: the user samples it and can be said to know it. A service draws fresh coins each time it is invoked. Writing R_{Service} for the coins of a single session, membership asserts that the predicate is satisfied for some coins the service could supply:

$$\mathcal{L}_{\mathcal{US}} \;=\; \left\{ (\mathsf{id}, \Phi) \;\middle|\; \exists\, \mathsf{data},\, R_{\mathsf{U}} : \begin{array}{c} \Phi\big(\mathsf{data},\, \mathsf{out}_{\mathsf{U}}(\mathsf{Prot}_{\mathsf{id}}\langle \mathsf{Service}_{\mathsf{id}}(\mathsf{sec}_{\mathsf{id}};\, R_{\mathsf{Service}}),\, \\ \mathsf{U}(\mathsf{data};\, R_{\mathsf{U}})\rangle)\big) = 1 \end{array} \right\}.$$

Membership so read is a statement about knowledge alone: satisfying coins exist, but nothing says a run produced them. That gap is the execution requirement of Section 2.1, which the next subsection imposes on top of membership. We drop the tapes from the notation in the rest of the paper.

Interface: A zkPoPE scheme exposes its parties to the environment $\mathcal{Z}$ through four input messages:

- $(\mathsf{sid}, \texttt{Setup}, (\mathsf{id}, \mathsf{Prot}, \mathsf{sec}))$: service registration, submitted by a service.
- $(\mathsf{sid}, \texttt{Setup}, \{(\mathsf{pid}_i, \mathsf{id}_i, \mathsf{Prot}_i)\}_{i=1}^{l})$: client-side setup, declaring the set of services a non-service party expects to interact with.
- $(\mathsf{sid}, \texttt{Prove}, \mathsf{id}, \Phi, \mathsf{data}, R_{\mathsf{U}})$: proving, submitted by the user; R_{U} is the user-side random tape discussed in Remark 3.1.
- $(\mathsf{sid}, \texttt{Verify}, \mathsf{id}, P_i, \Phi, \pi)$: non-interactive verification, submitted by the verifier.

The four tags are the four input handlers of $\mathcal{F}_{\mathsf{zkPoPE}}$ below.

3.2 Ideal Functionality

We capture the security requirements of zkPoPE through the ideal functionality $\mathcal{F}_{\mathsf{zkPoPE}}$ defined in Figure 2. $\mathcal{F}_{\mathsf{zkPoPE}}$ runs in the environment $\mathcal{Z}$, in the presence of a simulator Sim, and a set of machines $\mathsf{P} = \{P_1, \ldots, P_N\}$ partitioned into users (which we group together with the verifier and any auxiliary parties a concrete realization may require) and services. It is parameterized by the security parameter 1^λ and by statically chosen sets FC of fully corrupted and HCC of honest-but-curious corrupted parties.
Notation. Inside the handlers we use the following semantic instructions:

- **Break**: abort the current activation without output.

[10] Reclaim's verifier reconstructs part of the transcript and evaluates Φ on it, which restricts the predicates it supports; Section 5.2 fixes that restriction. It is a feature of how Reclaim verifies rather than of zkPoPE: a realization whose verifier checks a proof instead of re-evaluating the predicate would carry a different restriction, or none.

Functionality $\mathcal{F}_{\mathsf{zkPoPE}}$

On input $(\mathsf{sid}, \mathtt{Setup}, (\mathsf{id}, \mathsf{Prot}, \mathsf{sec}))$ from a service P:

1. Check there was no user-side `Setup` for sid, Break otherwise.
2. Check P is not in the universe of services $\mathcal{US}_{\mathsf{sid}}$, and there is no service with id id in the universe, Break otherwise.
3. Add P as $\mathsf{Service}_{\mathsf{id}} = (\mathsf{id}, \mathsf{sec}, \mathsf{Prot})$ to the universe of services for sid:
$$\mathcal{US}_{\mathsf{sid}} \leftarrow \mathcal{US}_{\mathsf{sid}} \cup (\mathsf{pid}, \mathsf{Service}_{\mathsf{id}}).$$
4. Send $(\mathsf{sid}, \mathtt{ServiceAdded}, \mathsf{pid}, \mathsf{id}, \mathsf{Prot})$ to Sim.

On input $(\mathsf{sid}, \mathtt{Setup}, \{\mathsf{pid}_i, \mathsf{id}_i, \mathsf{Prot}_i\}_{i=1}^{l})$ from honest users:

1. Check there was no user-side `Setup` for sid, Break otherwise.
2. Check that all input services are consistent with $\mathcal{US}_{\mathsf{sid}}$.
3. Send $(\mathsf{sid}, \mathtt{Setup})$ to Sim, receive in reply crs and a tuple
$$\mathsf{Algs} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}).$$
4. Store $(\mathsf{sid}, \mathsf{crs}, \mathsf{Algs})$.
5. Send $(\mathsf{sid}, \mathtt{Setup}, \mathsf{crs})$ to the calling parties.

On input $(\mathsf{sid}, \mathtt{Prove}, \mathsf{id}, \Phi, \mathsf{data}, R_{\mathsf{U}})$ from a party P:

1. Check there was `Setup` for sid.
2. Retrieve $(\mathsf{sid}, \mathsf{crs}, \mathsf{Algs})$ and $(\mathsf{pid}, \mathsf{Service}_{\mathsf{id}} = (\mathsf{id}, \mathsf{sec}, \mathsf{Prot})) \in \mathcal{US}_{\mathsf{sid}}$.
3. If P is honest and $\Phi(\mathsf{data}, \mathsf{reply}) = 0$ for every reply, send $(\mathsf{sid}, \mathtt{Prove}, \perp)$ to P and Break.
Simulate the protocol between user and service.
4. Compute $(\mathsf{reply}, \mathsf{out}_{\mathsf{Service}}) \leftarrow \mathsf{Prot}_{\mathsf{id}}\langle \mathsf{Service}_{\mathsf{id}}(\mathsf{sec}_{\mathsf{id}}; R_{\mathsf{Service}}), \mathsf{U}(\mathsf{data}; R_{\mathsf{U}})\rangle$, with R_{Service} the coins $\mathsf{Service}_{\mathsf{id}}$'s tape supplies for this activation.
5. Send $(\mathsf{sid}, \mathtt{Executed}, \mathsf{reply})$ to P and $(\mathsf{sid}, \mathtt{Executed}, \mathsf{out}_{\mathsf{Service}})$ to P_{pid}.
6. If P is corrupted, send $(\mathsf{sid}, \mathtt{Predicate}, P)$ to Sim, receive $(\mathsf{sid}, \mathtt{Predicate}, \Phi')$ in reply, and set $\Phi \leftarrow \Phi'$.
7. Compute the binary value $res = \Phi(\mathsf{data}, \mathsf{reply})$.
Issue a proof if the predicate holds.
8. Send $(\mathsf{sid}, \mathtt{Prove}, P, \Phi, \mathsf{id}, res, |\mathsf{reply}|)$ to Sim.
9. If res is equal to 0, send $(\mathsf{sid}, \mathtt{Prove}, \perp)$ to P, and Break.
10. Wait for a reply $(\mathtt{Proof}, \pi)$ from Sim.
11. Check $\mathsf{Verify}(\mathsf{crs}, \pi) = 1$, otherwise send $(\mathsf{sid}, \mathtt{Prove}, \perp)$ to P, and Break.
12. Send $(\mathsf{sid}, \mathtt{Prove}, P, \Phi, \mathsf{id}, \pi)$ to P.
13. Store $(P, \Phi, \mathsf{id}, \pi)$ to list Π_{sid}, and Break.

On input $(\mathsf{sid}, \mathtt{Verify}, \mathsf{id}, P_i, \Phi, \pi)$ from P:

1. Check there was `Setup` for sid.
2. Check if there is an entry $(P_i, \Phi, \mathsf{id}, \pi)$ in Π_{sid}:
   - If so, send $(\mathsf{sid}, \mathtt{Verify}, (\mathsf{id}, P_i, \Phi, \pi), 1)$ to P,
   - Otherwise, send $(\mathsf{sid}, \mathtt{Verify}, (\mathsf{id}, P_i, \Phi, \pi), 0)$ to P,

   and Break.

Figure 2: Ideal Functionality $\mathcal{F}_{\mathsf{zkPoPE}}$ for zkPoPE.

- **Parse**: extract well-formed messages matching a given pattern.

Every message $\mathcal{F}_{\mathsf{zkPoPE}}$ sends to a party is a *private delayed output* in the sense of [Can01]: Sim is told that a message to that party is pending and decides when it is delivered; its contents reach Sim only when the recipient is corrupted. We write $\mathsf{Algs} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for the bundle of scheme algorithms supplied by Sim at setup time, together with the common reference string crs; inside $\mathcal{F}_{\mathsf{zkPoPE}}$ these have their standard NIZK meanings, the protocol-level Verify checking a proof against crs. Sim produces crs by its own simulated setup and keeps the trapdoor, so $\mathcal{F}_{\mathsf{zkPoPE}}$ holds none. On a successful `Prove` activation $\mathcal{F}_{\mathsf{zkPoPE}}$ obtains the proof from Sim, sending it $(\mathsf{sid}, \texttt{Prove}, P, \Phi, \mathsf{id}, res, |\mathsf{reply}|)$ and awaiting a reply, as in the ideal functionality for non-interactive zero-knowledge of [GOS06]. The message is sent on every activation and not only on a satisfied predicate, since a party running a `Prove` session is observable whatever the predicate evaluates to, and it carries the length of the reply, as a channel carrying the reply would leak it; $\mathcal{F}_{\mathsf{zkPoPE}}$ awaits a proof only when $res = 1$. A corrupted prover fixes its predicate after the run, through Sim, and may submit $\Phi = \bot$ on input: such a prover holds the witness $(\mathsf{data}, \mathsf{reply})$ once the run is over and could evaluate any predicate on it, so nothing false becomes provable and the honest prover's guarantees are unaffected. An honest prover whose input no reply can complete to a satisfying pair is turned away before the run and without notice to Sim, as in the functionality of [GOS06], which ignores a `Prove` on $(x, w) \notin R$; the test is part of Φ's specification.

3.3 Relation to Existing Models

DECO [ZMM+20] introduced the oracle ideal functionality $\mathcal{F}_{\mathsf{Oracle}}$, a UC abstraction of a TLS query–response interaction with three roles, prover, verifier and server, and a query template instantiated by a private prover input. The verifier participates online, so non-interactive proofs are not covered; the definition binds the server side to standard TLS (Definition 3.1 in [ZMM+20]); and no proof object exists, since the designated verifier receives the evaluated statement $\mathsf{Stmt}(R)$ from the functionality and nothing else, so attestations are not transferable and verification is not a separate operation. Its descendants keep this shape. Garble-then-Prove [XYWY23] defines $\mathcal{F}_{\mathsf{AuthData}}$ on top of it, with the three roles, the query template and the commit-then-prove flow. ACTS [MVVZ26] keeps DECO's design and replaces its final zero-knowledge stage with a predicate blind signature, so that attestations are standard RSA-PSS signatures; its formal analysis is confined to that component. ORIGO [ELW+24] defines $\mathcal{F}_{DP}$ for a proxy verifier in a malicious-network setting, with the protocol fixed to TLS 1.3 and AES-GCM, which ties its theorem to that AEAD. DiStefano [CDH+25] rejects monolithic DECO-style proofs, since any change of primitive or ciphersuite forces a new one, and analyzes its handshake, query and commitment phases separately around the *designated-commitment TLS* primitive; this restores modularity but gives up any single object against which schemes could be compared, and stays inside the transport, with phases specific to TLS 1.3 and a record layer targeting AES-GCM. Ticket to Hide [LRV26] builds a bespoke functionality for proving a connection to one of several servers without revealing which; TLSNotary [KK24] is documented informally; Janus [LEFS23] formalizes asymmetric privacy through an HVZK proof system; Town Crier [ZCC+16] casts security as datagram authenticity for a feed delivered by an SGX-resident TLS client.

In every one of these works TLS is essential to the definition or the analysis, and no functionality in the line has been reused unchanged by a successor: each modified, re-tailored or abandoned its predecessor's, which is the incomparability of Section 1 playing out in the literature. $\mathcal{F}_{\mathsf{zkPoPE}}$ takes the other exit from protocol-inseparability: one functionality for the whole class, with the transport removed from it.

4 Distributed AEAD

The Reclaim Protocol relies on a three-party variant of authenticated encryption. The encryption and authentication of a message are jointly produced by two non-colluding parties, both distinct from the data party. The primitive must guarantee that no single party learns the inputs of the others, and that the data party alone cannot forge a valid ciphertext-tag pair. We define it directly over the class of stream-cipher AEAD

designs and do not pursue further generalization in this work. Concrete instantiations based on AES-GCM and ChaCha20-Poly1305, the AEAD schemes underlying TLS, appear in Appendix C.4.

4.1 Syntax

Fix associated spaces $\mathcal{K} = \mathcal{K}(\lambda)$ (keys), $\mathcal{N} = \mathcal{N}(\lambda)$ (nonces), $\mathcal{T} = \mathcal{T}(\lambda)$ (tags), $\mathcal{TM} = \mathcal{TM}(\lambda)$ (tag material), $\mathcal{AD} \subseteq \{0,1\}^*$ (associated data), $\mathcal{M} \subseteq \{0,1\}^*$ (messages), and $\mathcal{C} \subseteq \{0,1\}^*$ (ciphertexts).

Definition 4.1 (Distributed stream-cipher AEAD) A *distributed stream-cipher AEAD scheme* is a tuple

$$\mathsf{D\text{-}AEAD} = (\mathsf{KGen}, \mathsf{GenEncMat}, \mathsf{GenTagMat}, \mathsf{ComputeTag}, \mathsf{VerifyTag})$$

of PPT algorithms which are defined as follows:

- $\mathsf{KGen}(1^\lambda) \to k \in \mathcal{K}$;
- $\mathsf{GenEncMat} : \mathcal{K} \times \mathcal{N} \times \mathbb{N} \to \{0,1\}^*$ outputs an ℓ-bit keystream on (k, n, ℓ);
- $\mathsf{GenTagMat} : \mathcal{K} \times \mathcal{N} \to \mathcal{TM}$ outputs tag material $\boldsymbol{\tau} \in \mathcal{TM}$ on (k, n);
- $\mathsf{ComputeTag} : \mathcal{TM} \times \mathcal{AD} \times \mathcal{C} \to \mathcal{T}$ outputs a tag $T \in \mathcal{T}$ on $(\boldsymbol{\tau}, AD, C)$;
- $\mathsf{VerifyTag} : \mathcal{TM} \times \mathcal{AD} \times \mathcal{C} \times \mathcal{T} \to \{0,1\}$ outputs a bit on $(\boldsymbol{\tau}, AD, C, T)$, with $\mathsf{VerifyTag}(\boldsymbol{\tau}, AD, C, \mathsf{ComputeTag}(\boldsymbol{\tau}, AD, C)) = 1$ for every $(\boldsymbol{\tau}, AD, C) \in \mathcal{TM} \times \mathcal{AD} \times \mathcal{C}$.

The scheme defines two three-party interactive protocols Π_{AEnc} and Π_{ADec} (Figures 3–4), executed by the *key party* $\mathcal{P}_K$ holding k; the *data party* $\mathcal{P}_D$ holding message P and a *secret-index set* $\mathsf{SR} \subseteq [|P|]$ (designating positions of P whose contents must remain hidden from $\mathcal{P}_K$ and $\mathcal{P}_T$) in Π_{AEnc} and the ciphertext-tag pair (C, T) in Π_{ADec}; and the *tag party* $\mathcal{P}_T$ holding no private input.

We write $\mathsf{embed}_{\mathsf{SR},\ell}(M) \in \{0,1\}^\ell$ for the embedding of $M \in \{0,1\}^{|\mathsf{SR}|}$ at the positions indexed by SR, with zeros elsewhere.

Common input $(\mathsf{sid}, n, AD, |P|)$. Private inputs: $\mathcal{P}_K$: k; $\mathcal{P}_D$: P, SR; $\mathcal{P}_T$: $\perp$.

1. $\mathcal{P}_D$: sample $M \xleftarrow{\$} \{0,1\}^{|\mathsf{SR}|}$; set $\widetilde{P} \leftarrow P \oplus \mathsf{embed}_{\mathsf{SR},|P|}(M)$; send $\widetilde{P}$ to $\mathcal{P}_K$ and $\mathsf{embed}_{\mathsf{SR},|P|}(M)$ to $\mathcal{P}_T$.
2. $\mathcal{P}_K$: compute $K_s \leftarrow \mathsf{GenEncMat}(k, n, |P|)$ and $\boldsymbol{\tau} \leftarrow \mathsf{GenTagMat}(k, n)$; set $\widetilde{C} \leftarrow \widetilde{P} \oplus K_s$; send $(\widetilde{C}, \boldsymbol{\tau})$ to $\mathcal{P}_T$; output ok.
3. $\mathcal{P}_T$: set $C \leftarrow \widetilde{C} \oplus \mathsf{embed}_{\mathsf{SR},|P|}(M)$ and $T \leftarrow \mathsf{ComputeTag}(\boldsymbol{\tau}, AD, C)$; send (C, T) to $\mathcal{P}_D$; output ok.
4. $\mathcal{P}_D$: upon receipt of (C, T), output (C, T).

Figure 3: Π_{AEnc} protocol.

Common input (sid, n, AD). Private inputs: $\mathcal{P}_K$: k; $\mathcal{P}_D$: (C, T); $\mathcal{P}_T$: $\perp$.

1. $\mathcal{P}_D$: send (C, T) to $\mathcal{P}_T$.
2. $\mathcal{P}_T$: send $|C|$ to $\mathcal{P}_K$;
3. $\mathcal{P}_K$: compute $\boldsymbol{\tau} \leftarrow \mathsf{GenTagMat}(k, n)$; send $\boldsymbol{\tau}$ to $\mathcal{P}_T$.
4. $\mathcal{P}_T$: compute $T' \leftarrow \mathsf{ComputeTag}(\boldsymbol{\tau}, AD, C)$; **if** $T' \neq T$ send $\perp$ to $\mathcal{P}_K$ and $\mathcal{P}_D$ and output $\perp$; otherwise, send ok to $\mathcal{P}_K$ and $\mathcal{P}_D$ and output ok.
5. $\mathcal{P}_K$: upon ok from $\mathcal{P}_T$, compute $K_s \leftarrow \mathsf{GenEncMat}(k, n, |C|)$, send K_s to $\mathcal{P}_D$, and output ok; upon $\perp$ from $\mathcal{P}_T$, output $\perp$.
6. $\mathcal{P}_D$: upon K_s from $\mathcal{P}_K$, output $\widetilde{P} \leftarrow C \oplus K_s$; upon $\perp$ from $\mathcal{P}_T$, output $\perp$.

Figure 4: Π_{ADec} protocol.

Correctness. An honest execution of Π_{AEnc} delivers to $\mathcal{P}_D$ the pair (C, T) with $C = P \oplus \mathsf{GenEncMat}(k, n, |P|)$ and $T = \mathsf{ComputeTag}(\mathsf{GenTagMat}(k, n), AD, C)$, the AEAD record of P under (k, n); an honest execution of Π_{ADec} on (C, T) delivers P to $\mathcal{P}_D$ when T is the valid tag for (AD, C) under (k, n), and $\perp$ otherwise.

4.2 Security

We adopt a *static, single-party* corruption model. The adversary $\mathcal{A}$ is *semi-honest* when corrupting $\mathcal{P}_K$ or $\mathcal{P}_T$, observing the party's view but following the prescribed protocol, and *malicious* when corrupting $\mathcal{P}_D$, free to deviate arbitrarily from Π_{AEnc} and Π_{ADec}. This asymmetry is essential for Reclaim's setting, as discussed in Section 2.2.1.

We require two security properties of D-AEAD; Appendix C gives the games and the reductions to standard assumptions on the primitives. Integrity is stronger than standard AEAD INT-CTXT, since $\mathcal{P}_D$'s view contains the plaintext and adaptive protocol-internal state. Confidentiality is restricted to the SR positions: in Reclaim, the positions outside SR are the bytes the predicate discloses to the verifier, so their indistinguishability is not required.

Confidentiality. $\mathcal{P}_K$ or $\mathcal{P}_T$, semi-honestly running adaptively many concurrent honest sessions of Π_{AEnc} and Π_{ADec}, learns nothing about $\mathcal{P}_D$'s plaintext at the SR positions in Π_{AEnc} and nothing about the plaintext at any position in Π_{ADec}.

Ciphertext integrity. $\mathcal{P}_D$, arbitrarily deviating from Π_{AEnc} and Π_{ADec} across adaptively many concurrent sessions with honest $\mathcal{P}_K$ and $\mathcal{P}_T$, cannot produce a tuple (n, AD, C, T) that an honest run of Π_{ADec} accepts and that was not delivered to $\mathcal{P}_D$ in any prior Π_{AEnc} session.

Joint guarantee. Together the two properties say that every record an honest Π_{ADec} accepts was produced with knowledge of k, hence not by $\mathcal{P}_D$ alone, while the views of $\mathcal{P}_K$ and $\mathcal{P}_T$ reveal nothing about the plaintext at the SR positions in Π_{AEnc} and nothing about it at all in Π_{ADec}; and neither protocol completes without both $\mathcal{P}_K$ and $\mathcal{P}_T$. Reclaim uses this to obtain an authentic, redacted record of $\mathcal{P}_D$'s payloads held jointly by two independent parties.

5 The Reclaim Protocol

5.1 Hybrid model and interface

We specify the Reclaim Protocol $\mathcal{RP}$ in the $(\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{CERT}}, \mathcal{F}_{\mathsf{KE}}, \mathcal{F}_{\mathsf{US}})$-hybrid UC model [Can01], with A_K and A_T semi-honest. $\mathcal{F}_{\mathsf{SC}}$ [CK02] carries every message among $\{\mathsf{U}, \mathsf{A}_K, \mathsf{A}_T\}$, $\mathcal{F}_{\mathsf{KE}}$ [CK02] produces the Phase A session key, $\mathcal{F}_{\mathsf{CERT}}$ [Can04] provides three signing facilities, and $\mathcal{F}_{\mathsf{US}}$ models the U–S_{id} wire: every Phase A–C message sent to or received from S_{id} "over the network" below travels over $\mathcal{F}_{\mathsf{US}}$, whose leakage regime is motivated in Section 2.2.1. Appendix B.3 gives the message syntax of each functionality.

The three $\mathcal{F}_{\mathsf{CERT}}$ instances are pre-registered with fixed roles. $\mathcal{F}_{\mathsf{CERT}}[S_{\mathsf{id}}]$ models S_{id}'s long-term TLS certificate and authenticates its key-exchange contribution in Phase A, following [GMP+08b]; $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_K]$ and $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_T]$ sign the attestors' per-session transcript views in Phase D, and theirs are the only signatures $\mathcal{V}$ checks. All Prove sessions share the three instances and the three pairwise $\mathcal{F}_{\mathsf{SC}}$ channels $\mathsf{sid}_{\mathsf{UA}_K}$, $\mathsf{sid}_{\mathsf{UA}_T}$, $\mathsf{sid}_{\mathsf{A}_K\mathsf{A}_T}$; concurrent sessions are disambiguated by a per-session tag ssid, fixed in Phase A and carried by every later message of the session, which both Phase-D signatures cover.

Two routing decisions keep S_{id} unmodified while authentication of the TLS session resides with the attestors. Although A_K initiates the key exchange, the network endpoint S_{id} observes belongs to U: every non-$\mathcal{F}_{\mathsf{KE}}$ Phase A message between A_K and S_{id} is relayed by U over the network. And the call to $\mathcal{F}_{\mathsf{KE}}$ is made by A_K and S_{id} directly, with U not a party to it. S_{id}'s view is thus a standard TLS handshake with U, while the output of $\mathcal{F}_{\mathsf{KE}}$ and the verdict of $\mathcal{F}_{\mathsf{CERT}}[S_{\mathsf{id}}]$ are held by A_K and never appear on the network.

5.2 Admissible Protocols and Predicates

$\mathcal{RP}$ is analyzed for a restricted class of application-layer protocols and of predicates over them. This subsection fixes that class. We state the conditions informally here; their formal counterparts are Definitions B.1, B.2 and B.5 of Appendix B.2.

The protocol. We require $\mathsf{Prot}_{\mathsf{id}}$ to be *request–reply*: the user sends one message *Req*, computed from its input data and its coins by a map we write $\mathsf{Prot}_{\mathsf{id}}.\mathsf{U}_1$; the service answers with one message *Resp* computed from $\mathsf{sec}_{\mathsf{id}}$ and *Req*; and the user's reply is that response. We require further that $\mathsf{Prot}_{\mathsf{id}}$ be *request-extractable*: from a request, one can recover an input and coins that regenerate exactly that request.
The latter requirement is the price of a design choice: U is never asked to show how it built *Req* from data. Demanding that would admit every protocol, since the demonstration would itself yield the extraction, but it would put a proof about the request format on the user's critical path, with cost growing in the format and the predicate. $\mathcal{RP}$ keeps the user's cost low, so a malicious U may put an arbitrary string on the wire, and the request is the only handle the analysis has on the session; request extractability is what turns it back into an input. Little is lost, because $\mathsf{Prot}_{\mathsf{id}}.\mathsf{U}_1$ is public and, in the common cases, invertible.

- Where it is invertible the condition holds with no assumption beyond the shape of the protocol: requests that are the input verbatim, or a public serialization of fields of data, possibly with a nonce or timestamp drawn from the coins, whose parser is the inverse.
- Where it is not — a request carrying, for example, a signature under a U-held key, a commitment, or a zero-knowledge proof — an idealized model widens the class: a local random-oracle or algebraic-group analysis hands the extractor the adversary's oracle queries or group-element representations, which for many such formats is what it needs. The widening is partial.

Appendix B.2 treats the notion in full; widening the class further we leave to future work.

The predicate. A predicate Φ is a per-session parameter agreed between U and $\mathcal{V}$ in advance. $\mathcal{F}_{\mathsf{zkPoPE}}$ evaluates it on the user's input and the reply; $\mathcal{V}$ holds neither data nor $\mathsf{sec}_{\mathsf{id}}$ and evaluates only on the request and response bytes it reconstructs. We therefore require Φ to be *admissible* for $\mathsf{Prot}_{\mathsf{id}}$ (Definition B.5): there exists a predicate $\widehat{\Phi}$ over request–response pairs that agrees with Φ on every execution, and no rejection reply satisfies $\widehat{\Phi}$. The first condition makes $\mathcal{V}$'s verdict the one the language speaks about; the second makes $\mathcal{V}$ reject the proof of a session whose request no data generates.
We require moreover that $\widehat{\Phi}$ have the following form. Its *format* names the map $\mathsf{Prot}_{\mathsf{id}}.\mathsf{U}_1$, specialized to the request layout in question, together with the lengths $|Req|, |Resp|$ and two position sets $\mathsf{SR}_{\mathsf{req}} \subseteq \{1, \ldots, |Req|\}$ and $\mathsf{SR}_{\mathsf{resp}} \subseteq \{1, \ldots, |Resp|\}$ marking the bytes hidden from $\mathcal{V}$ and from both attestors; both sets depend on $\widehat{\Phi}$ and $\mathsf{Prot}_{\mathsf{id}}$ alone. Since $\mathsf{Prot}_{\mathsf{id}}$ has fluid but deterministic structure, the format is a small program rather than a constant table. Its *condition* is a propositional combination over $\{\wedge, \vee, \neg\}$ of atoms "the byte at position p equals the constant c", with p outside $\mathsf{SR}_{\mathsf{req}}$ on the request side and outside $\mathsf{SR}_{\mathsf{resp}}$ on the response side, given as an offset or as a parse-tree location under $\mathsf{Prot}_{\mathsf{id}}$; bytes are not interpreted beyond constant equality and not compared with one another. We write $\widehat{\Phi}(Req, Resp) \in \{0, 1\}$ for the result; since no atom reads a withheld position, it is the same on the zero-filled pair $\mathcal{V}$ reconstructs. Rejection soundness then amounts to the response-side atoms being inconsistent with the response head returned on a malformed request. We call predicates of this shape *equivalence predicates*.
Finally, $\widehat{\Phi}$ must *determine what it discloses*: any two pairs satisfying it agree at every position outside $\mathsf{SR}_{\mathsf{req}}$ and $\mathsf{SR}_{\mathsf{resp}}$. $\mathcal{V}$ receives every such byte, so otherwise a proof could reveal which of two satisfying transcripts took place, which the statement (id, Φ) does not. Consequently the disclosed bytes are a function of Φ and $\mathsf{Prot}_{\mathsf{id}}$, and a predicate of this form is fixed by its position sets and the bytes it discloses; we take that pair as its representation wherever predicates are compared, as in $\mathcal{F}_{\mathsf{zkPoPE}}$'s record. Disjunction and negation remain available where they admit no second value at a disclosed position. An admissible predicate of this form we call *supported*.
The description of $\widehat{\Phi}$ is that of Φ: the format and the condition are what U and $\mathcal{V}$ agree on, and Φ is the same condition read on the user's input and reply. We therefore write Φ for both and let the arguments disambiguate; the appendix keeps them apart.

Notation and conventions. For a position set $\mathsf{R} \subseteq \{1, \ldots, \ell\}$ and $M \in \{0,1\}^{|\mathsf{R}|}$, $\mathsf{embed}_{\mathsf{R},\ell}(M) \in \{0,1\}^{\ell}$ places the i-th symbol of M at the i-th smallest position of R and zero elsewhere, so $X \mapsto X \oplus \mathsf{embed}_{\mathsf{R},\ell}(M)$ is the identity outside R and, for uniform M, randomizes X on R; $\mathsf{zero}_{\mathsf{R}}(X)$ is X with the positions in R set to zero. The algorithms $\mathsf{GenEncMat}$, $\mathsf{GenTagMat}$, $\mathsf{ComputeTag}$, $\mathsf{VerifyTag}$ are those of $\mathsf{D\text{-}AEAD}$ (Definition 4.1),

instantiated for the session's cipher suite as in Appendix C.4; we omit their associated-data argument, which $\mathcal{RP}$ does not rely on. KDF [Kra10] expands the $\mathcal{F}_{\mathsf{KE}}$ key κ into two directional keys and two initialization vectors $(k_{\mathsf{enc}}, k_{\mathsf{dec}}, IV_{\mathsf{enc}}, IV_{\mathsf{dec}})$, the vectors serving as the D-AEAD nonces of the request and response records. We describe $\mathcal{RP}$ for one AEAD record per direction; several records are handled by deriving each record's nonce from the direction's IV and the record sequence number, as in TLS 1.3 [Res18].

5.3 Protocol $\mathcal{RP}$

Figure 5 in Appendix A summarizes the message flow described below.

Session state. All Prove sessions share the three $\mathcal{F}_{\mathsf{SC}}$ channels, so U, A_K and A_T each keep, per sid, a table of active sessions indexed by ssid, recording the session's position in Phases A–D and every value generated or received in it. A_K creates the entry when it generates ssid in Phase A; A_T on the first Phase-B message, from U or A_K, bearing a new ssid; U on the $(\mathsf{sid}, \mathsf{Prove}, \ldots)$ input, binding it to ssid on receipt of $(\mathsf{handshake\text{-}done}, \mathsf{ssid})$ after checking $\mathsf{ssid} = (\mathsf{sid}, \mathsf{U}, \mathsf{id}, r)$ against the $\mathsf{sid}_{\mathsf{KE}}$ it relayed. A party processes a message under ssid only if it is the one its state for ssid expects next from that sender on that channel, and drops it silently otherwise.

Setup. On $(\mathsf{sid}, \mathsf{Setup}, (\mathsf{id}, \mathsf{Prot}, \mathsf{sec}))$ from $\mathcal{Z}$, S_{id} records $(\mathsf{id}, \mathsf{Prot}, \mathsf{sec})$ as its service descriptor for sid and spawns $\mathcal{F}_{\mathsf{CERT}}[S_{\mathsf{id}}]$ by issuing $(\mathsf{Sign}, \mathsf{sid}_{S_{\mathsf{id}}}, \bot)$ and discarding the response. On $(\mathsf{sid}, \mathsf{Setup}, \{(\mathsf{pid}_i, \mathsf{id}_i, \mathsf{Prot}_i)\}_{i=1}^{l})$ from $\mathcal{Z}$, each of U, A_K, A_T, $\mathcal{V}$ records the list as its admissible services for sid; A_K and A_T spawn $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_K]$ and $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_T]$ in the same way, and the three channels $\mathsf{sid}_{\mathsf{UA}_K}, \mathsf{sid}_{\mathsf{UA}_T}, \mathsf{sid}_{\mathsf{A}_K\mathsf{A}_T}$ are established via $\mathcal{F}_{\mathsf{SC}}$.

Protocol $\mathcal{RP}$: Prove

Only U *is driven by* $\mathcal{Z}$*;* A_K*,* A_T *and* S_{id} *react to the messages of Phases A–D, and every party silently drops a message that fails its checks or the session-state convention.* On input $(\mathsf{sid}, \mathsf{Prove}, \mathsf{id}, \Phi, \mathsf{data}, R_{\mathsf{U}})$ from $\mathcal{Z}$, U:

1. If id is not in its admissible-service list, outputs $(\mathsf{sid}, \mathsf{Prove}, \bot)$ and halts; otherwise retrieves $(\mathsf{pid}_{S_{\mathsf{id}}}, \mathsf{Prot})$.
2. Computes $\mathit{Req} \leftarrow \mathsf{Prot}_{\mathsf{id}}.\mathsf{U}_1(\mathsf{data}; R_{\mathsf{U}})$ under the format named by Φ and extracts $\mathsf{SR}_{\mathsf{req}}$, $\mathsf{SR}_{\mathsf{resp}}$. If *Req* differs at a position outside $\mathsf{SR}_{\mathsf{req}}$ from the value Φ fixes there, so that no response can satisfy Φ, outputs $(\mathsf{sid}, \mathsf{Prove}, \bot)$ and halts.
3. Runs Phases A–D. On recovering *Resp* at the end of Phase C, outputs $(\mathsf{sid}, \mathsf{Executed}, \mathit{Resp})$ and evaluates $\Phi(\mathsf{data}, \mathit{Resp})$; if 0, outputs $(\mathsf{sid}, \mathsf{Prove}, \bot)$ and halts without entering Phase D.
4. If a phase aborts, outputs $(\mathsf{sid}, \mathsf{Prove}, \bot)$; on completing Phase D with transcript π, outputs $(\mathsf{sid}, \mathsf{Prove}, \mathsf{U}, \Phi, \mathsf{id}, \pi)$.

Phase A — Handshake ***(participants*** **U,** A_K**,** S_{id}***)***

- On the Prove input, U sends id to A_K via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{UA}_K}]$.
- A_K checks id against its list, retrieving $(\mathsf{pid}_{S_{\mathsf{id}}}, \mathsf{Prot})$, and aborts otherwise; samples $r \xleftarrow{\$} \{0,1\}^{\lambda}$; sets $\mathsf{sid}_{\mathsf{KE}} \leftarrow (\mathsf{A}_K, \mathsf{pid}_{S_{\mathsf{id}}}, \mathsf{id}, r)$ and $\mathsf{ssid} \leftarrow (\mathsf{sid}, \mathsf{U}, \mathsf{id}, r)$; and sends $(\mathsf{HandshakeRequest}, \mathsf{sid}_{\mathsf{KE}})$ to U, who forwards it to S_{id} over the network.
- On $(\mathsf{HandshakeRequest}, \mathsf{sid}_{\mathsf{KE}})$ with $\mathsf{sid}_{\mathsf{KE}} = (\mathsf{A}_K, \mathsf{pid}_{S_{\mathsf{id}}}, \mathsf{id}, r)$ matching its PID and a registered id, S_{id} obtains $\sigma_{S_{\mathsf{id}}}$ on $(\mathsf{sid}_{\mathsf{KE}}, \mathsf{A}_K, S_{\mathsf{id}})$ from $\mathcal{F}_{\mathsf{CERT}}[S_{\mathsf{id}}]$, sends it to U over the network, and sends $(\mathsf{Establish\text{-}session}, \mathsf{sid}_{\mathsf{KE}}, \mathsf{pid}_{S_{\mathsf{id}}}, \mathsf{A}_K, \mathsf{responder})$ to $\mathcal{F}_{\mathsf{KE}}$. U forwards $\sigma_{S_{\mathsf{id}}}$ to A_K.
- A_K verifies $\sigma_{S_{\mathsf{id}}}$ on $(\mathsf{sid}_{\mathsf{KE}}, \mathsf{A}_K, S_{\mathsf{id}})$ at $\mathcal{F}_{\mathsf{CERT}}[S_{\mathsf{id}}]$, aborting on 0, and sends $(\mathsf{Establish\text{-}session}, \mathsf{sid}_{\mathsf{KE}}, \mathsf{A}_K, \mathsf{pid}_{S_{\mathsf{id}}}, \mathsf{initiator})$ to $\mathcal{F}_{\mathsf{KE}}$.
- On $(\mathsf{key}, \mathsf{sid}_{\mathsf{KE}}, \kappa)$ from $\mathcal{F}_{\mathsf{KE}}$, A_K and S_{id} each derive $(k_{\mathsf{enc}}, k_{\mathsf{dec}}, IV_{\mathsf{enc}}, IV_{\mathsf{dec}}) \leftarrow \mathsf{KDF}(\kappa, \mathsf{sid}_{\mathsf{KE}})$. A_K sends $(\mathsf{handshake\text{-}done}, \mathsf{ssid})$ to U, who records ssid.

Phase B — Request Handling ***(participants*** U***,*** A_K***,*** A_T***, and*** S_{id} ***as a TLS endpoint; instantiates*** $\mathsf{D\text{-}AEAD}.\Pi_{\mathsf{AEnc}}$***)***

- On $(\mathsf{handshake\text{-}done}, \mathsf{ssid})$, U samples $M_{\mathsf{req}} \xleftarrow{\$} \{0,1\}^{|\mathsf{SR}_{\mathsf{req}}|}$, sets $\widetilde{Req} \leftarrow Req \oplus \mathsf{embed}_{\mathsf{SR}_{\mathsf{req}},|Req|}(M_{\mathsf{req}})$, and sends $(\mathsf{ssid}, \widetilde{Req}, \mathsf{SR}_{\mathsf{req}})$ to A_K via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{UA}_K}]$ and $(\mathsf{ssid}, \mathsf{embed}_{\mathsf{SR}_{\mathsf{req}},|Req|}(M_{\mathsf{req}}))$ to A_T via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{UA}_T}]$.
- On $(\mathsf{ssid}, \widetilde{Req}, \mathsf{SR}_{\mathsf{req}})$, A_K records $\mathsf{SR}_{\mathsf{req}}$, computes $K_{\mathsf{req}} \leftarrow \mathsf{GenEncMat}(k_{\mathsf{enc}}, IV_{\mathsf{enc}}, |Req|)$, $\tau_{\mathsf{req}} \leftarrow \mathsf{GenTagMat}(k_{\mathsf{enc}}, IV_{\mathsf{enc}})$ and $\widetilde{Req}_{\mathsf{enc}} \leftarrow \widetilde{Req} \oplus K_{\mathsf{req}}$, and sends $(\mathsf{ssid}, \widetilde{Req}_{\mathsf{enc}}, \tau_{\mathsf{req}}, \mathsf{SR}_{\mathsf{req}})$ to A_T via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{A}_K\mathsf{A}_T}]$.
- On (ssid, Y) from U and $(\mathsf{ssid}, \widetilde{Req}_{\mathsf{enc}}, \tau_{\mathsf{req}}, \mathsf{SR}_{\mathsf{req}})$ from A_K, A_T checks that $|Y| = |\widetilde{Req}_{\mathsf{enc}}|$ and that Y is zero outside $\mathsf{SR}_{\mathsf{req}}$, aborting otherwise; records $(\mathsf{ssid}, \mathsf{SR}_{\mathsf{req}})$; computes $Req_{\mathsf{enc}} \leftarrow \widetilde{Req}_{\mathsf{enc}} \oplus Y$ and $T_{\mathsf{req}} \leftarrow \mathsf{ComputeTag}(\tau_{\mathsf{req}}, Req_{\mathsf{enc}})$; and sends $(\mathsf{ssid}, Req_{\mathsf{enc}}, T_{\mathsf{req}})$ to U via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{UA}_T}]$.
- U forwards $(Req_{\mathsf{enc}}, T_{\mathsf{req}})$ to S_{id} over the network; ssid never travels on the wire.
- S_{id} checks $\mathsf{VerifyTag}(\mathsf{GenTagMat}(k_{\mathsf{enc}}, IV_{\mathsf{enc}}), Req_{\mathsf{enc}}, T_{\mathsf{req}}) = 1$, aborting otherwise; recovers $Req \leftarrow Req_{\mathsf{enc}} \oplus \mathsf{GenEncMat}(k_{\mathsf{enc}}, IV_{\mathsf{enc}}, |Req_{\mathsf{enc}}|)$; computes $(Resp, \mathsf{out}_{\mathsf{Service}}) \leftarrow \mathsf{Prot}_{\mathsf{id}}.\mathsf{Service}(\mathsf{sec}, Req; R_{\mathsf{Service}})$ on fresh coins; outputs $(\mathsf{sid}, \mathsf{Executed}, \mathsf{out}_{\mathsf{Service}})$ to $\mathcal{Z}$; and sends $(Resp_{\mathsf{enc}}, T_{\mathsf{resp}})$ to U over the network, with $Resp_{\mathsf{enc}} \leftarrow Resp \oplus \mathsf{GenEncMat}(k_{\mathsf{dec}}, IV_{\mathsf{dec}}, |Resp|)$ and $T_{\mathsf{resp}} \leftarrow \mathsf{ComputeTag}(\mathsf{GenTagMat}(k_{\mathsf{dec}}, IV_{\mathsf{dec}}), Resp_{\mathsf{enc}})$. This is standard TLS record processing; S_{id} runs no code of $\mathcal{RP}$.

Phase C — Response Handling ***(participants*** U***,*** A_T***,*** A_K***; instantiates*** $\mathsf{D\text{-}AEAD}.\Pi_{\mathsf{ADec}}$***)***

- On $(Resp_{\mathsf{enc}}, T_{\mathsf{resp}})$ from the network, U sends $(\mathsf{ssid}, Resp_{\mathsf{enc}}, T_{\mathsf{resp}})$ to A_T via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{UA}_T}]$; A_T sends $(\mathsf{ssid}, |Resp_{\mathsf{enc}}|)$ to A_K via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{A}_K\mathsf{A}_T}]$; A_K replies with $(\mathsf{ssid}, \tau_{\mathsf{resp}})$, $\tau_{\mathsf{resp}} \leftarrow \mathsf{GenTagMat}(k_{\mathsf{dec}}, IV_{\mathsf{dec}})$.
- A_T computes $b \leftarrow \mathsf{VerifyTag}(\tau_{\mathsf{resp}}, Resp_{\mathsf{enc}}, T_{\mathsf{resp}})$ and sends $(\mathsf{ssid}, \mathsf{ok})$ if $b = 1$, and $(\mathsf{ssid}, \perp)$ otherwise, to both A_K and U, outputting the same.
- On $(\mathsf{ssid}, \mathsf{ok})$, A_K computes $K_{\mathsf{resp}} \leftarrow \mathsf{GenEncMat}(k_{\mathsf{dec}}, IV_{\mathsf{dec}}, |Resp_{\mathsf{enc}}|)$ and sends $(\mathsf{ssid}, K_{\mathsf{resp}})$ to U via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{UA}_K}]$; on $(\mathsf{ssid}, \perp)$ it outputs $\perp$.
- On $(\mathsf{ssid}, \mathsf{ok})$ from A_T and $(\mathsf{ssid}, K_{\mathsf{resp}})$ from A_K, U outputs $Resp \leftarrow Resp_{\mathsf{enc}} \oplus K_{\mathsf{resp}}$; on $(\mathsf{ssid}, \perp)$ it outputs $\perp$.

Phase D — Transcript Generation ***(participants*** U***,*** A_K***,*** A_T***)***

- On completing Phase C, U sends $(\mathsf{begin\text{-}transcript}, \mathsf{ssid}, \mathsf{SR}_{\mathsf{resp}})$ to A_K via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{UA}_K}]$.
- A_K retrieves the $\mathsf{SR}_{\mathsf{req}}$ recorded for ssid, computes $Req_{\mathsf{null}} \leftarrow \mathsf{zero}_{\mathsf{SR}_{\mathsf{req}}}\left(\widetilde{Req}\right)$ and $\widetilde{K}_{\mathsf{resp}} \leftarrow \mathsf{zero}_{\mathsf{SR}_{\mathsf{resp}}}(K_{\mathsf{resp}})$, obtains σ_K on $(\mathsf{U}, \mathsf{id}, \mathsf{ssid}, \mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}}, Req_{\mathsf{null}}, \widetilde{K}_{\mathsf{resp}})$ from $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_K]$, and sends $(\mathsf{begin\text{-}transcript}, \mathsf{ssid}, \mathsf{id}, \mathsf{SR}_{\mathsf{resp}})$ to A_T via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{A}_K\mathsf{A}_T}]$ and $(\mathsf{ssid}, \sigma_K, Req_{\mathsf{null}}, \widetilde{K}_{\mathsf{resp}})$ to U via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{UA}_K}]$.
- A_T retrieves the $\mathsf{SR}_{\mathsf{req}}$ recorded for ssid, computes $\widetilde{Resp}_{\mathsf{enc}} \leftarrow \mathsf{zero}_{\mathsf{SR}_{\mathsf{resp}}}(Resp_{\mathsf{enc}})$, obtains σ_T on $(\mathsf{U}, \mathsf{id}, \mathsf{ssid}, \mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}}, \widetilde{Resp}_{\mathsf{enc}})$ from $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_T]$, and sends $(\mathsf{ssid}, \sigma_T, \widetilde{Resp}_{\mathsf{enc}})$ to U via $\mathcal{F}_{\mathsf{SC}}[\mathsf{sid}_{\mathsf{UA}_T}]$.
- U assembles $\pi \leftarrow (\sigma_K, \sigma_T, \mathsf{id}, \mathsf{ssid}, \mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}}, Req_{\mathsf{null}}, \widetilde{K}_{\mathsf{resp}}, \widetilde{Resp}_{\mathsf{enc}})$ and outputs $(\mathsf{sid}, \mathsf{Prove}, \mathsf{U}, \Phi, \mathsf{id}, \pi)$ to $\mathcal{Z}$.

Protocol $\mathcal{RP}$**:** Verify ***(non-interactive;*** $\mathcal{V}$ ***holds*** A_K***,*** A_T ***and*** sid_K***,*** sid_T ***as protocol parameters)***

On input $(\mathsf{sid}, \mathsf{Verify}, \mathsf{id}, P_i, \Phi, \pi)$ from $\mathcal{Z}$, $\mathcal{V}$ outputs $(\mathsf{sid}, \mathsf{Verify}, (\mathsf{id}, P_i, \Phi, \pi), b)$, where $b = 0$ if check 1 or 2 fails and $b = b_1 \wedge b_2 \wedge b_3$ otherwise.

1. id is in its admissible-service list for sid.
2. π parses as $(\sigma_K, \sigma_T, \mathsf{id}_\pi, \mathsf{ssid}, \mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}}, Req_{\mathsf{null}}, \widetilde{K}_{\mathsf{resp}}, \widetilde{Resp}_{\mathsf{enc}})$ with $\mathsf{ssid} = (\mathsf{sid}', P', \mathsf{id}', r)$, $\mathsf{id}_\pi = \mathsf{id}' = \mathsf{id}$, $\mathsf{sid}' = \mathsf{sid}$, $P' = P_i$, and $(\mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}})$ the position sets fixed by the format of Φ.
3. b_1 and b_2 are the verdicts of $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_K]$ on σ_K over $(P_i, \mathsf{id}, \mathsf{ssid}, \mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}}, Req_{\mathsf{null}}, \widetilde{K}_{\mathsf{resp}})$ and of $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_T]$ on σ_T over $(P_i, \mathsf{id}, \mathsf{ssid}, \mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}}, \widetilde{Resp}_{\mathsf{enc}})$.
4. $b_3 \leftarrow \Phi(Req_{\mathsf{null}}, \widehat{Resp})$ with $\widehat{Resp} \leftarrow \widetilde{Resp}_{\mathsf{enc}} \oplus \widetilde{K}_{\mathsf{resp}}$.

5.4 Security

Theorem 5.1 Let D-AEAD be a distributed stream-cipher AEAD scheme satisfying the confidentiality and ciphertext-integrity guarantees of Section 4.2, let every $\mathsf{Prot}_{\mathsf{id}}$ in the deployment be request–reply and request-extractable, and let every predicate be supported for its protocol, in the sense of Section 5.2. Then the Reclaim protocol $\mathcal{RP}$ of Section 5 UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{zkPoPE}}$ of Section 3 in the $(\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{KE}}, \mathcal{F}_{\mathsf{CERT}}, \mathcal{F}_{\mathsf{US}})$-hybrid model, under the admissible corruption pattern recalled in Section 5.4.1.

5.4.1 Model and Corruption Pattern

$\mathcal{F}_{\mathsf{US}}$ and the admissible corruption pattern are those of Section 2.2.1. The pattern has four cases, which the proof treats separately: Case A (passive), $FC = HCC = \emptyset$; Case B (malicious U), $FC = \{\mathsf{U}\}$, $HCC = \emptyset$; Case C, $HCC = \{\mathsf{A}_K\}$; and Case D, $HCC = \{\mathsf{A}_T\}$, with $FC = \emptyset$ in the last two. Phases B and C of $\mathcal{RP}$ instantiate D-AEAD's Π_{AEnc} and Π_{ADec} under $\mathsf{U} \leftrightarrow \mathcal{P}_D$, $\mathsf{A}_K \leftrightarrow \mathcal{P}_K$, $\mathsf{A}_T \leftrightarrow \mathcal{P}_T$, with $\mathsf{SR}_{\mathsf{req}}$ and $\mathsf{SR}_{\mathsf{resp}}$ as the secret-index sets. At most one of the three is corrupted in any execution, so Case B is D-AEAD's malicious-$\mathcal{P}_D$ regime and Cases C, D its semi-honest $\mathcal{P}_K$ and $\mathcal{P}_T$ regimes.

5.4.2 Simulator

Sim runs the hybrid functionalities and an in-head copy of every uncorrupted party, forwarding a semi-honest attestor's view to $\mathcal{A}$. On $\mathcal{F}_{\mathsf{zkPoPE}}$.Setup it commits the algorithms (Setup, Prove, Verify) of Section 5, with Verify checking σ_K, σ_T under the verification algorithms it commits to $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_K], \mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_T]$. It releases each delayed output of $\mathcal{F}_{\mathsf{zkPoPE}}$ when the corresponding party of $\mathcal{RP}$ would produce it.

Honest User (Cases A, C, D). $\mathcal{F}_{\mathsf{zkPoPE}}$'s announcement $(P, \Phi, \mathsf{id}, res, |\mathsf{reply}|)$ tells Sim that a session has begun. A supported Φ fixes, through its format and its disclosure-determinacy, the position sets $(\mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}})$, the lengths, and the bytes at every disclosed position; let $Req^\star, Resp^\star$ carry those bytes and zeros on the two sets. Sim runs $\mathcal{RP}$ with its in-head parties on the request $Req^\star$ and, in place of the service's computation, the response $Resp^\star$ (a zero string of the announced length when $res = 0$), stopping after Phase C when $res = 0$. The proof it returns is the π its in-head U assembles. The simulation differs from a real execution only in the plaintexts, which agree with the real ones outside the secret positions whenever anything derived from them is signed.

Malicious User (Case B). $\mathcal{A}$ drives U's channels; the in-head attestors and service respond as prescribed. When the in-head S_{id} accepts a request record, Sim recovers the request $Req' = \widetilde{Req}^{\mathcal{A}} \oplus Y^{\mathcal{A}}$ from $\mathcal{A}$'s two Phase-B messages — the same reconstruction the honest A_T performs, whose check that $Y^{\mathcal{A}}$ is zero outside $\mathsf{SR}^{\mathcal{A}}_{\mathsf{req}}$ makes Req' agree with the signed Req_{null} at every disclosed position — and applies the extractor to obtain an input and tape $(\mathsf{data}', R'_{\mathsf{U}})$ regenerating Req'. It submits these to $\mathcal{F}_{\mathsf{zkPoPE}}$.Prove on U's behalf, receives the reply as the corrupted caller's output, encrypts it as the response record, and runs Phases C and D against $\mathcal{A}$. It then names the predicate: the supported one with $\mathcal{A}$'s position sets asserting the disclosed bytes of the transcript, or the identically-false one if the reply is a rejection. On $res = 1$ it answers the proof request with the π its attestors signed, so that $\mathcal{F}_{\mathsf{zkPoPE}}$'s record matches what $\mathcal{A}$ holds.

5.4.3 Indistinguishability

Zero-knowledge (Cases A, C, D). The simulation replaces the request and response, on the positions in $\mathsf{SR}_{\mathsf{req}}$ and $\mathsf{SR}_{\mathsf{resp}}$ (and everywhere in a response that fails Φ, which is never signed), by fixed bytes. In Case A, $\mathcal{A}$ controls the wire, and ciphertext integrity lets the simulated S_{id} and A_T accept only each other's records. In Case D, $\mathcal{F}_{\mathsf{US}}$ delivers every record unaltered, and $\mathcal{A}$'s view is $\mathcal{P}_T$'s view of Π_{AEnc} and Π_{ADec}, which contains the records $\mathcal{A}$ sees on the wire in Case A. In both cases a hybrid over sessions and directions reduces the change to D-AEAD's $\mathcal{P}_T$-confidentiality (IND-CCA-$\mathcal{P}_T$). In Case C the wire is hidden by $\mathcal{F}_{\mathsf{US}}$ and $\mathcal{A}$'s view is $\mathcal{P}_K$'s: everything in it is a function of the session key and of $\widetilde{Req}$, which is one-time-padded on $\mathsf{SR}_{\mathsf{req}}$ and equal to the public bytes elsewhere, so the change is invisible (IND-CPA-$\mathcal{P}_K$, advantage 0).

Soundness (Case B). $\mathcal{F}_{\mathsf{zkPoPE}}.\mathtt{Verify}$ accepts only proofs it recorded, so indistinguishability from the real execution says that a malicious U obtains no accepted π beyond those, and that those are accepted. Three hops. First, every record that S_{id} or A_T accepts is one the other honest side produced in that session, by $\mathsf{D\text{-}AEAD}$ ciphertext integrity. Second, $\mathcal{F}_{\mathsf{CERT}}$'s sign tables pin an accepted π^* to a single session: both signatures cover the session tag, the position sets and the redacted views, and a mismatch between the sessions behind σ_K^* and σ_T^* needs a nonce collision, of probability at most $q^2 2^{-\lambda-1}$ over q sessions. $\mathcal{V}$ then evaluates Φ on the disclosed bytes of that session's transcript, and by disclosure-determinacy exactly one supported predicate with those position sets accepts it — the one Sim names. Third, the extractor's input regenerates the session's request, so $\mathcal{F}_{\mathsf{zkPoPE}}$'s in-head run returns the response $\mathcal{A}$ received and records π under that same predicate; this costs the extraction error over q sessions. The second hop does not use the corruption of U: with an honest U the session's predicate is the one it was given, so in Cases A, C and D as well $\mathcal{V}$ accepts the proofs $\mathcal{F}_{\mathsf{zkPoPE}}$ recorded and no others, up to the same collision term.

6 Benchmarks

Setup. We benchmark the Reclaim Protocol with the prover running on a mobile phone equipped with a Qualcomm Snapdragon 888 5G processor and 8 GB of RAM, released in 2021, representative of the lower-mid range of current consumer mobile hardware. The two TEE attestors run on Google Cloud confidential-computing virtual machines in the `europe-north1-a` region, backed by AMD SEV-SNP. The prover communicates with the attestors over Wi-Fi with a typical round-trip time of 75 ms to the region. Every reported value is the mean of ten independent executions, and we record the wall-clock interval between the start of the TLS handshake on the prover and the moment the verifier accepts the resulting attestation.

Scope of the measurement. End-to-end latency in our protocol is dominated by communication between the prover and the attestors, and this cost scales with the length of the server response rather than with the size of the proven substring. A modest implementation-level optimization that ships only the proven bytes in a small number of network rounds saves approximately 0.5 s, which is immaterial at the timescales reported below. We therefore fix the benchmark to the most demanding configuration and prove the *entire* server response in every run.

Comparison with prior work. Existing zkTLS protocols are not evaluated on response sizes comparable to ours. The largest published evaluation is Janus [LEFS23], which proves 8 kB of sensitive TLS data in 6.2 s under TLS 1.2 and 6.7 s under TLS 1.3. All other known constructions evaluate at smaller sizes, with most below 4 kB and none above the 8 kB ceiling set by Janus. Since we prove the entire response, our protocol attests to approximately 500 kB of data in the same time budget, roughly 60 times larger. A direct comparison at the sizes of Table 1 is therefore not possible, and we instead place each row next to the

Table 1: End-to-end latency of the Reclaim Protocol against a plain HTTPS fetch of the same resource, executed with `curl` on a lower-mid range mobile device. Each value is the mean of ten runs. Sizes report the HTTPS response in binary kilobytes (1 kB = 1024 B). Δ is the difference of the two timings and isolates the cost of the protocol.

Site	Size (kB)	TLS (s)	Reclaim (s)	Δ (s)
`example.com`	1.23	0.39	2.90	2.51
`iana.org`	6.00	0.62	3.11	2.49
`jquery.com`	25.12	0.45	3.21	2.76
`wikipedia.org`	117.50	0.52	4.67	4.15
`wikipedia.org/Main_Page`	225.35	0.60	4.79	4.19
`wikipedia.org/Software_eng.`	408.73	0.64	5.67	5.03
`github.com`	557.39	0.68	6.66	5.98
`wikipedia.org/World_War_II`	1201.66	0.84	7.28	6.44

wall-clock time of a plain HTTPS fetch of the same resource on the same device, executed with `curl`. This isolates the protocol overhead from the irreducible network and transport cost.

Results. Table 1 reports the timings. Across response sizes spanning three orders of magnitude, from 1.23 kB to 1201.66 kB, the protocol completes in under 7.3 s. The additive overhead over the HTTPS baseline grows from 2.5 s to 6.4 s, a 2.5-fold increase across a roughly 1000-fold increase in response size. These absolute timings, reached at proven data sizes up to two orders of magnitude beyond any previously reported, easily place the Reclaim Protocol within deployable range on lower-mid range mobile hardware.

Acknowledgments

We would like to thank Martin Albrecht, Benjamin Dowling, Aleksei Ermishkin, Madhavan Malolan, Andy Rupp, and Adhiraj Singh for helpful discussions and suggestions. We also thank the anonymous reviewers for helpful comments.

Use of AI tools. We used Anthropic's Claude (Fable 5.1) to proofread the technical details and proofs, check the text for consistency, improve its style, readability and editorial quality, and composing TikZ schemes.

References

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 505–514. ACM Press, May / June 2014.

[Ber05] Daniel J. Bernstein. The poly1305-aes message-authentication code. In *Proceedings of the 12th International Conference on Fast Software Encryption*, FSE'05, page 32–49, Berlin, Heidelberg, 2005. Springer-Verlag.

[BP15] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 236–261. Springer, Berlin, Heidelberg, November / December 2015.

[BRS24] Annabelle Backman, Justin Richer, and Manu Sporny. HTTP Message Signatures. RFC 9421, February 2024.

[BSCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology – CRYPTO 2013*, volume 8043 of *LNCS*, pages 90–108. Springer, 2013.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE, 2001.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 219–233. IEEE, 2004.

[CDG+18] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 280–312. Springer, Cham, April / May 2018.

[CDH+25] Sofía Celi, Alex Davidson, Hamed Haddadi, Gonçalo Pestana, and Joe Rowell. DiStefano: Decentralized infrastructure for sharing trusted encrypted facts and nothing more. In *Network and Distributed System Security Symposium (NDSS) 2025*, 2025.

[CGCD+17] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy*, pages 451–466, 2017.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 597–608. ACM Press, November 2014.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351. Springer, 2002.

[Con26] Deirdre Connolly. ML-KEM Post-Quantum Key Agreement for TLS 1.3. Internet-Draft draft-ietf-tls-mlkem-07, Internet Engineering Task Force, February 2026. Work in Progress.

[CSB+26] Jalen Chuang, Alex Seto, Nicolas Berrios, Stephan van Schaik, Christina Garman, and Daniel Genkin. Tee.fail: Breaking trusted execution environments via ddr5 memory bus interposition. In *47th IEEE Symposium on Security and Privacy (IEEE S&P '26)*. IEEE Computer Society, 2026.

[Dam92] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 445–456. Springer, Berlin, Heidelberg, August 1992.

[Dwo07] Morris J. Dworkin. Recommendation for block cipher modes of operation: Galois/Counter mode (GCM) and GMAC. NIST Special Publication 800-38D, National Institute of Standards and Technology, Gaithersburg, MD, November 2007.

[ELW+24] Jens Ernstberger, Jan Lauinger, Yinnan Wu, Arthur Gervais, and Sebastian Steinhorst. ORIGO: Proving provenance of sensitive data with constant communication. Cryptology ePrint Archive, Paper 2024/447, 2024.

[FK11] Sheila Frankel and Suresh Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, February 2011.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62. Springer, Cham, August 2018.

[FNR22] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP/1.1. RFC 9112, June 2022.

[GMP+08a] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of tls. In *International Conference on Provable Security*, pages 313–327. Springer, 2008.

[GMP+08b] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of TLS. In *Provable Security (ProvSec 2008)*, volume 5324 of *LNCS*, pages 313–327. Springer, 2008.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer, Berlin, Heidelberg, May / June 2006.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 99–108. ACM Press, June 2011.

[Int15] Intel Corporation. *Intel® Software Guard Extensions (Intel® SGX)*, 2015. Official Intel overview document.

[IT21] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[KK24] Maciej Kalka and Marek Kirejczyk. A comprehensive review of tlsnotary protocol, 2024.

[KMO+15] Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. (de-) constructing tls 1.3. In *International Conference on Cryptology in India*, pages 85–102. Springer, 2015.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *LNCS*, pages 631–648. Springer, 2010.

[LEFS23] Jan Lauinger, Jens Ernstberger, Andreas Finkenzeller, and Sebastian Steinhorst. Janus: Fast privacy-preserving data provenance for TLS. Cryptology ePrint Archive, Paper 2023/1377, 2023.

[LR25] Anna Lysyanskaya and Leah Namisa Rosenbloom. Adaptive uc nizk for practical applications. In *International Conference on Cryptology and Information Security in Latin America*, pages 76–108. Springer, 2025.

[LRV26] Ryan Little, Daniel S. Roche, and Mayank Varia. Ticket to hide: Private, practical proofs of provenance for TLS. Cryptology ePrint Archive, Paper 2026/540, 2026.

[MV04] David A. McGrew and John Viega. The security and performance of the galois/counter mode (gcm) of operation. In *Proceedings of the 5th International Conference on Cryptology in India*, INDOCRYPT'04, page 343–355, Berlin, Heidelberg, 2004. Springer-Verlag.

[MVVZ26] Pierpaolo Della Monica, Ivan Visconti, Andrea Vitaletti, and Marco Zecchini. ACTS: Attestations of contents in TLS sessions, 2026. Cryptology ePrint Archive, Paper 2026/755.

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, Berlin, Heidelberg, August 2003.

[NL15] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539, May 2015.

[Not22] Mark Nottingham. Building Protocols with HTTP. RFC 9205, June 2022. BCP 56.

[Pro25] Reclaim Protocol. Reclaim Protocol Whitepaper (v2), 2025.

[RD08] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[Res18] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[Str] Stripe, Inc. Resolve webhook signature verification errors. https://docs.stripe.com/webhooks/signature.

[VBMW+18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, page 991–1008, USA, 2018. USENIX Association.

[vSKGY20] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. SGAxe: How SGX fails in practice. https://sgaxeattack.com/, 2020.

[XYWY23] Xiang Xie, Kang Yang, Xiao Wang, and Yu Yu. Lightweight authentication of web data via garble-then-prove. Cryptology ePrint Archive, Paper 2023/964, 2023.

[ZCC+16] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 270–282, New York, NY, USA, 2016. Association for Computing Machinery.

[ZMM+20] Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. Deco: Liberating web data using decentralized oracles for tls. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1919–1938. ACM, October 2020.

A Message Flow of the Reclaim Protocol

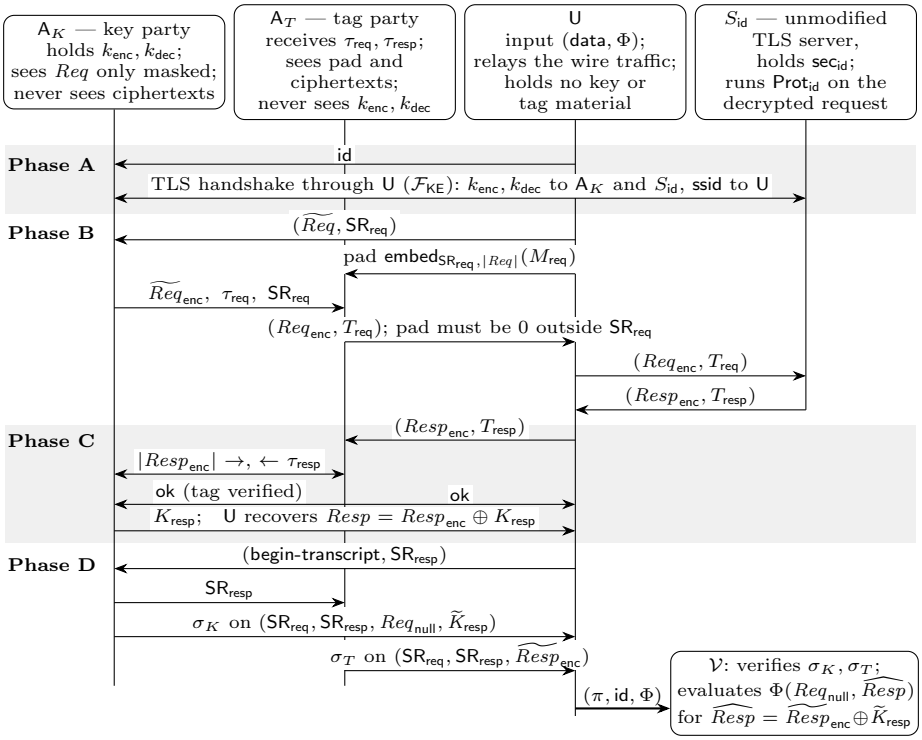

Figure 5: Message flow of the Reclaim protocol $\mathcal{RP}$: what each party holds and the messages of Phases A–D (Section 5.3). Messages among U, A_K and A_T travel over secure channels $\mathcal{F}_{\mathsf{SC}}$; those between U and S_{id} over the public network as ordinary TLS records. The traffic keys live only at A_K, the ciphertexts and the User's one-time pad only at A_T, so neither view alone recovers the session data. The proof $\pi = (\sigma_K, \sigma_T, \mathsf{id}, \mathsf{ssid}, \mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}}, Req_{\mathsf{null}}, \widetilde{K}_{\mathsf{resp}}, \widetilde{Resp}_{\mathsf{enc}})$ carries the disclosed positions and the request and response with private bytes zeroed, both signed by the attestors.

B Preliminaries

B.1 Notation

We write n for the security parameter and 1^n for its unary encoding; PPT abbreviates probabilistic polynomial-time, and $\mathsf{negl}(n)$ denotes a negligible function. For $n \in \mathbb{N}$ we write $[n] := \{1, \ldots, n\}$. We use $|\mathcal{X}|$ for the cardinality of a set and $|x|$ for the bit-length of a string x; $\perp$ denotes a distinguished failure ("no value") symbol.

We write $x \xleftarrow{\$} \mathcal{X}$ for sampling x uniformly from a finite set $\mathcal{X}$, and $y \leftarrow A(x)$ for assigning to y the output of a (possibly randomized) algorithm A on input x. To make randomness explicit we write $y \leftarrow A(x; \rho)$ for running A on input x with random tape ρ, and $(y, \rho) \leftarrow A(x)$ for saving the random tape used into ρ; with ρ fixed, A is deterministic, which lets a relation or a proof refer to the exact coins an algorithm used. For a probabilistic experiment we write $\Pr[\, E : \mathsf{exp} \,]$ for the probability of event E after running exp. Two distribution ensembles X, Y are *identically distributed*, written $X \equiv Y$, if they coincide, and *computationally indistinguishable*, written $X \approx_c Y$, if no PPT distinguisher separates them with non-negligible advantage.

For a two-party protocol Π we write

$$\Pi\langle P_1(\mathsf{in}_1),\ P_2(\mathsf{in}_2)\rangle$$

for the joint execution in which party P_i is initialized with private input in_i. We write $\mathsf{out}_{P_i}\big(\Pi\langle P_1(\mathsf{in}_1),\ P_2(\mathsf{in}_2)\rangle\big)$ for the local output produced by P_i at the end of the execution, and collect both outputs as

$$(\mathsf{out}_{P_1},\ \mathsf{out}_{P_2}) \leftarrow \Pi\langle P_1(\mathsf{in}_1),\ P_2(\mathsf{in}_2)\rangle.$$

When the execution is randomized, each party may additionally receive a random tape; we write $\Pi\langle P_1(\mathsf{in}_1;\ r_1),\ P_2(\mathsf{in}_2;\ r_2)\rangle$ to make those tapes explicit.

B.2 Request–Reply Protocols and Request Extractability

We define the class of two-party protocols we call *request–reply*, a notion of *request extractability* for protocols in that class, and a matching notion of *admissibility* for predicates over them. Both concern a protocol and a predicate alone. Section 5.2 explains informally why the analysis of $\mathcal{RP}$ appeals to them.

The request–reply class. We treat protocols carrying one message in each direction.

Definition B.1 (Request–reply protocol) A two-party protocol Prot between U and $\mathsf{Service}$ is *request–reply* if it is specified by two efficiently computable maps

$$\mathsf{Prot.U}_1 : (\mathsf{data},\ R_{\mathsf{U}}) \mapsto \mathit{Req},$$
$$\mathsf{Prot.Service} : (\mathsf{sec},\ \mathit{Req},\ R_{\mathsf{Service}}) \mapsto (\mathit{Resp},\ \mathsf{out}_{\mathsf{Service}}),$$

and its execution consists of U computing and sending $\mathit{Req} \leftarrow \mathsf{Prot.U}_1(\mathsf{data};\ R_{\mathsf{U}})$; $\mathsf{Service}$ computing $(\mathit{Resp},\ \mathsf{out}_{\mathsf{Service}}) \leftarrow \mathsf{Prot.Service}(\mathsf{sec},\ \mathit{Req};\ R_{\mathsf{Service}})$, sending Resp to U, and locally outputting $\mathsf{out}_{\mathsf{Service}}$; and U locally outputting the response it received, $\mathsf{reply} = \mathit{Resp}$. We write

$$\mathcal{R}_{\mathsf{Prot}} \ = \ \big\{\, \mathsf{Prot.U}_1(\mathsf{data};\ R) \ :\ \mathsf{data} \in \{0,1\}^*,\ R \in \{0,1\}^* \,\big\}$$

for the set of *well-formed* requests of Prot, and posit as part of Prot's specification a publicly recognizable set $\mathcal{E}_{\mathsf{Prot}} \subseteq \{0,1\}^*$ of *rejection replies*, such that for every $\mathit{Req} \notin \mathcal{R}_{\mathsf{Prot}}$ and every $(\mathsf{sec}, R_{\mathsf{Service}})$ the service replies with some $\mathit{Resp} \in \mathcal{E}_{\mathsf{Prot}}$ and produces the null local output $\mathsf{out}_{\mathsf{Service}} = \bot$.

Taking the user's output to be the response verbatim costs nothing over the predicates admitted below (Remark B.6). The rejection set formalizes the assumption of Section 2.1 that the Service rejects inputs of incorrect format; that it is publicly recognizable makes a rejection reply as informative as no reply to any party inspecting the transcript. Over HTTP, the `4xx` status line in the response head identifies a malformed request to a party holding neither data nor sec.

Request extractability. The notion asks that a request identify an input producing it, in the following sense.

Definition B.2 (Request extractability) A request–reply protocol Prot is *request-extractable* if for every PPT algorithm $\mathcal{A}$ there exists a PPT algorithm $\mathsf{E}_{\mathcal{A}}$ such that, for every auxiliary input $\mathsf{aux} \in \{0,1\}^{\mathrm{poly}(n)}$,

$$\varepsilon^{\mathsf{ext}}_{\mathsf{Prot},\mathcal{A}}(n) \ :=\ \Pr\big[\, \mathit{Req}^* \in \mathcal{R}_{\mathsf{Prot}} \ \wedge\ \mathsf{Prot.U}_1(\mathsf{data}';\ R') \neq \mathit{Req}^* \,\big] \ \leq\ \mathsf{negl}(n),$$

where $\varepsilon^{\mathsf{ext}}_{\mathsf{Prot},\mathcal{A}}(n)$ denotes the supremum over aux of the probability, taken over the experiment

$$R_{\mathcal{A}} \xleftarrow{\$} \{0,1\}^{\mathrm{poly}(n)};$$
$$\mathit{Req}^* \leftarrow \mathcal{A}(1^n,\ \mathsf{aux};\ R_{\mathcal{A}});$$
$$(\mathsf{data}', R') \leftarrow \mathsf{E}_{\mathcal{A}}(1^n,\ \mathsf{aux},\ R_{\mathcal{A}},\ \mathit{Req}^*).$$

The extractor receives $\mathcal{A}$'s auxiliary input and random tape and is invoked once, on the request $\mathcal{A}$ has produced: it is *straight-line*, never rewinding $\mathcal{A}$ or restarting it on fresh coins. This is the most demanding of the usual extractor shapes, and the one our application requires. The extractor need only produce *some* preimage of Req^*, not the one $\mathcal{A}$ had in mind; the two may differ whenever data carries components the request does not transmit — fields the format discards, credentials that authenticate without appearing verbatim, slack in an encoding. Nothing is lost: by Definition B.1 everything after the first message is a function of Req^* and the service's state, so any two preimages elicit the same response and, by Definition B.5 below, the same predicate value.
The following unconditional variant is the form in which the protocols surveyed at the end of this subsection meet the condition.

Definition B.3 (Request invertibility) A request–reply protocol Prot is *request-invertible* if there is a single deterministic polynomial-time algorithm E such that $\mathsf{Prot.U}_1(\mathsf{data}'; R') = Req$ for every $Req \in \mathcal{R}_{\mathsf{Prot}}$, where $(\mathsf{data}', R') \leftarrow \mathsf{E}(Req)$.

Fact B.4 Request invertibility implies request extractability with $\varepsilon^{\mathsf{ext}}_{\mathsf{Prot},\mathcal{A}}(n) = 0$ for every $\mathcal{A}$: take $\mathsf{E}_{\mathcal{A}} = \mathsf{E}$, which ignores $(\mathsf{aux}, R_{\mathcal{A}})$.

Admissible predicates. Request extractability recovers an input regenerating the request; it does not relate a predicate's value on that input to its value on the transcript. The second notion does.

Definition B.5 (Admissible predicate) Let Prot be a request–reply protocol. A predicate Φ over input–reply pairs of Prot is *admissible for* Prot if there exists a predicate $\widehat{\Phi}$ over request–response pairs such that:

(i) *(Transcript determinacy.)* For all data, R_{U} and $Resp$,

$$\Phi(\mathsf{data},\, Resp) \;=\; \widehat{\Phi}\big(\mathsf{Prot.U}_1(\mathsf{data};\, R_{\mathsf{U}}),\, Resp\big).$$

(ii) *(Rejection soundness.)* $\widehat{\Phi}(Req,\, Resp) = 0$ for every Req and every $Resp \in \mathcal{E}_{\mathsf{Prot}}$.

Clause (i) says that Φ depends on data only through the request the execution puts on the wire; clause (ii) rules out the predicates a malformed request could satisfy. The equivalence predicates of Section 5.2 are specified from the transcript side: one writes down $\widehat{\Phi}$, and Φ is that condition read on the user's input and reply. Clause (i) is then immediate, and clause (ii) holds whenever the response-side atoms are inconsistent with a `4xx` response head.

Remark B.6 (Dropping the user's post-processing) Definition B.1 fixes $\mathsf{reply} = Resp$. One might allow the user a post-processing map $\mathsf{Prot.U}_2$, taking $(\mathsf{data}, R_{\mathsf{U}}, Resp)$ to reply — decrypting under a key held in data, say. Over admissible predicates this is no restriction. Clause (i) would then read

$$\Phi\big(\mathsf{data},\, \mathsf{Prot.U}_2(\mathsf{data},\, R_{\mathsf{U}},\, Resp)\big) \;=\; \widehat{\Phi}\big(\mathsf{Prot.U}_1(\mathsf{data};\, R_{\mathsf{U}}),\, Resp\big),$$

so the predicate's value depends on nothing $\mathsf{Prot.U}_2$ contributes beyond the transcript, and replacing $\mathsf{Prot.U}_2$ by the identity, with $\widehat{\Phi}$ as the predicate, leaves requests, responses and predicate values unchanged on every input. This is a property of the class treated here, not of the notion of Section 3.1, where the user's output is an arbitrary local output of an arbitrary two-party protocol.

The two notions combine as follows.

Lemma 1 (Extraction consistency) Let Prot be request-extractable, let Φ be admissible for Prot, let $\mathcal{A}$ be PPT with extractor $\mathsf{E}_{\mathcal{A}}$, and fix $(\mathsf{sec}, R_{\mathsf{Service}})$. Let $Req^* \leftarrow \mathcal{A}(1^n, \mathsf{aux}; R_{\mathcal{A}})$, let $(\mathsf{data}', R') \leftarrow \mathsf{E}_{\mathcal{A}}(1^n, \mathsf{aux}, R_{\mathcal{A}}, Req^*)$, and let $(Resp^*, \mathsf{out}^*_{\mathsf{Service}}) \leftarrow \mathsf{Prot.Service}(\mathsf{sec},\, Req^*;\, R_{\mathsf{Service}})$. Then, except with probability $\varepsilon^{\mathsf{ext}}_{\mathsf{Prot},\mathcal{A}}(n)$, exactly one of the following holds.

(i) $Req^* \in \mathcal{R}_{\mathsf{Prot}}$, and the execution $\mathsf{Prot}\langle\mathsf{Service}(\mathsf{sec};\, R_{\mathsf{Service}}),\, \mathsf{U}(\mathsf{data}';\, R')\rangle$ places exactly Req^* on the wire. It therefore runs to $(Resp^*, \mathsf{out}^*_{\mathsf{Service}})$, and its user-side output satisfies $\Phi(\mathsf{data}',\, Resp^*) = \widehat{\Phi}(Req^*,\, Resp^*)$.

(ii) $Req^* \notin \mathcal{R}_{\mathsf{Prot}}$, and $\widehat{\Phi}(Req^*, Resp^*) = 0$.

Proof. The two cases are exhaustive and mutually exclusive by definition of $\mathcal{R}_{\mathsf{Prot}}$. In case (i), Definition B.2 gives $\mathsf{Prot.U}_1(\mathsf{data}'; R') = Req^*$ except with probability $\varepsilon^{\mathsf{ext}}_{\mathsf{Prot},\mathcal{A}}(n)$; the first message of the execution is this value by Definition B.1, and the service's computation is a function of $(\mathsf{sec}, Req^*, R_{\mathsf{Service}})$ alone, so it returns $(Resp^*, \mathsf{out}^*_{\mathsf{Service}})$. The claim about Φ is clause (i) of Definition B.5 applied to $(\mathsf{data}', R', Resp^*)$. In case (ii), Definition B.1 gives $Resp^* \in \mathcal{E}_{\mathsf{Prot}}$, and clause (ii) of Definition B.5 gives $\widehat{\Phi}(Req^*, Resp^*) = 0$.

Achievability. Request extractability concerns Prot alone. It holds trivially when $\mathsf{data} = Req$ and $\mathsf{Prot.U}_1$ is the identity, the shape of the oracle functionality of DECO [ZMM+20] and its descendants, and it holds by Fact B.4 whenever $\mathsf{Prot.U}_1$ is a public injective serialization, the extractor being its parser. An HTTP request is of this kind: the message syntax of [FNR22] fixes the encoding of request line, header fields and body and specifies the parse recovering each, so a $\mathsf{Prot.U}_1$ assembling a request from fields of data is inverted by that parse. Invertibility survives a nonce, timestamp or padding drawn from R_{U}: the parser reads them off Req and returns them as R', which is where letting the extractor choose R' does its work.
Where Req carries a value the party producing it cannot open — a signature under a U-held key, a commitment, a zero-knowledge proof, for example — $\mathsf{Prot.U}_1$ is not efficiently invertible, and Definition B.2 asks instead for an extractor that reads the signing key or the opening off the adversary's code and coins. Invertibility is a property of the function $\mathsf{Prot.U}_1$, settled by examining Prot; extractability asserts an extractor for *every* adversary, in the $\forall\mathcal{A}\,\exists\mathsf{E}_{\mathcal{A}}$ shape of knowledge-of-exponent [Dam92] and of the extraction assumptions behind succinct arguments, and like them it is not falsifiable in the sense of [Nao03, GW11]. Discharging it means assuming it, or working in a model that hands the extractor a lever on the adversary: its random-oracle queries, or the representations an algebraic adversary supplies with its group elements [FKL18]. The lever helps only where the adversary itself produced the values to be recovered, so the widening is partial.[11]
Quantifying over every auxiliary input has a consequence worth stating. Let $\mathcal{D}$ be the adversary that outputs its auxiliary input. Its extractor $\mathsf{E}_{\mathcal{D}}$ must, for every $\mathsf{aux} \in \mathcal{R}_{\mathsf{Prot}}$, return a preimage of aux except with negligible probability over its own coins; it is a randomized inverter of $\mathsf{Prot.U}_1$ on $\mathcal{R}_{\mathsf{Prot}}$. In the plain model, request extractability is therefore request invertibility up to a negligible error, and the non-invertible formats above are not extractable at all: the auxiliary input may carry a signed request whose signing key no extractor recovers. This is the barrier of [BCPR14, BP15] in our setting. The notion has content beyond invertibility only in idealized models in which a well-formed request cannot be handed over but must be built with oracle queries: a local random oracle, whose queries the extractor sees, or an algebraic group, whose representations it is given. The same observation is what lets a simulation-based proof use the notion. Against the dummy adversary, whose requests are its auxiliary input, the extractor is $\mathsf{E}_{\mathcal{D}}$, fixed once, and the simulator runs it on the environment's instructions without knowing the environment. We adopt Definition B.2 as stated so as to cover these models uniformly; the invertible cases above need none of this.
Section 2.1 raises the complementary question of the protocol–predicate pairs for which knowledge of $(\mathsf{data}, \mathsf{reply})$ already witnesses execution. Both are conditions on (Prot, Φ); a systematic account is left to future work (Section 2.3).

B.3 Ideal Functionalities

We work in the $(\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{CERT}}, \mathcal{F}_{\mathsf{KE}}, \mathcal{F}_{\mathsf{US}})$-hybrid model. The first three are standard UC functionalities lifted from prior work [CK02, Can04], with one structural modification to $\mathcal{F}_{\mathsf{CERT}}$ (described below) that we adopt so that the simulator can commit to its certification algorithms at setup time rather than be queried per signature. $\mathcal{F}_{\mathsf{US}}$ is new to this work; it was motivated informally in the Technical Overview, and we give its formal definition here. All four functionalities are stated below. A semi-honestly corrupted attestor follows

[11] Which values the lever reaches depends on how the oracle is modeled. If it is local to the execution under analysis, every query put to it by or for the adversary is visible and extraction goes through. A *global* random oracle [CJS14, CDG+18] is shared with everything outside that execution, which may query it unobserved; the lever is then lost, and only the restricted observable and programmable variants recover part of it. A value an outside party obtained from the oracle and handed to $\mathcal{A}$ is, in this respect, auxiliary input.

its code, and the adversary obtains its view. $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{F}_{\mathsf{CERT}}$ and $\mathcal{F}_{\mathsf{KE}}$ treat such an attestor as uncorrupted: in their descriptions, *corrupted* means fully corrupted. In $\mathcal{F}_{\mathsf{US}}$, whose mode depends on the attestors alone, it means semi-honestly corrupted, the only corruption the admissible pattern allows an attestor.

Secure Channel $\mathcal{F}_{\mathsf{SC}}$

Functionality $\mathcal{F}_{\mathsf{SC}}$ (secure channel between two parties P_i, P_j).

Implicit state: per session identifier sid, an activation record (initially empty).

- **Activation.** On $(\mathsf{Establish\text{-}session}, \mathsf{sid}, P', \mathsf{role})$ from a party P, where $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: record (P, P', role) and notify Sim. Mark the session active once both P_i and P_j have submitted matching activations.
- **Transmission.** On $(\mathsf{Send}, \mathsf{sid}, m)$ from an active partner P: leak $(\mathsf{sid}, |m|, P)$ to Sim, then output $(\mathsf{Received}, \mathsf{sid}, m)$ to the other partner.
- **Deactivation.** On $(\mathsf{Expire\text{-}session}, \mathsf{sid})$ from an active partner: mark the session inactive and notify Sim.

Figure 6: Functionality $\mathcal{F}_{\mathsf{SC}}$ [CK02].

Guarantee. Between two uncorrupted partners, $\mathcal{F}_{\mathsf{SC}}$ leaks only the lengths of transmitted messages to Sim; in particular, $\mathcal{A}$ learns no contents of $(\mathsf{Send}, \cdot, m)$ messages, nor can it modify, reorder, or drop them.

Certification $\mathcal{F}_{\mathsf{CERT}}$

This is an adaptation of the UC certification functionality of [Can04]: the simulator commits a $(\mathsf{Sign}, \mathsf{Vfy})$ algorithm pair at the first activation of each instance, and $\mathcal{F}_{\mathsf{CERT}}$ subsequently runs these algorithms locally on Sign and Verify calls. Unforgeability when the owner is uncorrupted is enforced by $\mathcal{F}_{\mathsf{CERT}}$'s internal sign table $\mathcal{T}$, independently of the Vfy algorithm supplied by Sim.

Functionality $\mathcal{F}_{\mathsf{CERT}}[\mathsf{pid}]$ (certification, owned by signer identity pid; instances indexed by $\mathsf{sid} = (\mathsf{pid}, \mathsf{sid}')$).

State: an algorithm pair $(\mathsf{Sign}, \mathsf{Vfy})$, initially undefined; a table $\mathcal{T} \subseteq \{0,1\}^* \times \{0,1\}^*$ of signed pairs, initially empty.

- **Initialization** (triggered on the first activation under sid). Notify Sim of the activation; on receipt of $(\mathsf{Algorithms}, \mathsf{sid}, \mathsf{Sign}, \mathsf{Vfy})$ from Sim, record $(\mathsf{Sign}, \mathsf{Vfy})$ as this instance's algorithms. Then proceed with the current activation below.
- **Sign.** On $(\mathsf{Sign}, \mathsf{sid}, m)$ from pid: compute $\sigma \leftarrow \mathsf{Sign}(m)$, insert (m, σ) into $\mathcal{T}$, and output $(\mathsf{Signature}, \mathsf{sid}, m, \sigma)$ to pid. Notify Sim of the activation.
- **Verify.** On $(\mathsf{Verify}, \mathsf{sid}, m, \sigma)$ from any party P:
  - ◇ if pid is uncorrupted, output $(\mathsf{Verified}, \mathsf{sid}, m, 1)$ if $(m, \sigma) \in \mathcal{T}$, and $(\mathsf{Verified}, \mathsf{sid}, m, 0)$ otherwise;
  - ◇ if pid is corrupted, output $(\mathsf{Verified}, \mathsf{sid}, m, \mathsf{Vfy}(m, \sigma))$.

Figure 7: Functionality $\mathcal{F}_{\mathsf{CERT}}[\mathsf{pid}]$, adapted from [Can04] with simulator-provided algorithms.

Guarantee (unforgeability). If the owner pid is uncorrupted, $\mathcal{F}_{\mathsf{CERT}}[\mathsf{pid}].\mathsf{Verify}$ outputs 1 on (m, σ) if and only if pid previously requested $\mathcal{F}_{\mathsf{CERT}}[\mathsf{pid}].\mathsf{Sign}(m)$ and received σ in reply. In particular, no third party, including one that holds the committed Sign and Vfy algorithms via Sim's setup-time disclosure, can produce a fresh $(m^*, \sigma^*) \notin \mathcal{T}$ that $\mathcal{F}_{\mathsf{CERT}}[\mathsf{pid}]$ accepts.

Functionality $\mathcal{F}_{\mathsf{KE}}$ (key exchange between two parties P_i, P_j).

State: per session identifier sid, an activation record (initially empty).

- **Activation.** On $(\mathsf{Establish\text{-}session}, \mathsf{sid}, P_i, P_j, \mathsf{role})$ from a party $P \in \{P_i, P_j\}$, with $\mathsf{role} \in \{\mathsf{initiator}, \mathsf{responder}\}$: record $(P, P_i, P_j, \mathsf{role})$ and notify Sim.
- **Key delivery.** Once both P_i and P_j have submitted matching activations under sid:
  - ◇ if both endpoints are uncorrupted, sample $\kappa \xleftarrow{\$} \{0,1\}^\lambda$, output $(\mathsf{key}, \mathsf{sid}, \kappa)$ to each of P_i, P_j, and notify Sim of the delivery;
  - ◇ if either endpoint is corrupted, request κ from Sim and output $(\mathsf{key}, \mathsf{sid}, \kappa)$ to each of P_i, P_j.

Figure 8: Functionality $\mathcal{F}_{\mathsf{KE}}$ [CK02].

Key Exchange $\mathcal{F}_{\mathsf{KE}}$

Guarantee. When both endpoints are uncorrupted, κ is uniformly distributed in $\{0,1\}^\lambda$ and is delivered identically to both endpoints; Sim learns only the activation and not κ.

User–Service Channel $\mathcal{F}_{\mathsf{US}}$

The new functionality $\mathcal{F}_{\mathsf{US}}$ is parameterized by the corruption status of the attestors: it behaves as an adversarial network when A_K and A_T are both uncorrupted, and as a UC secure channel when either is corrupted. The motivation for this dichotomy is laid out in Subsection 2.2.1 of the Technical Overview.

Functionality $\mathcal{F}_{\mathsf{US}}$ (User–Service channel; transmission mode determined by the corruption status of A_K and A_T).

State: per session identifier sid, an activation record (initially empty).

- **Activation.** On $(\mathsf{Establish\text{-}session}, \mathsf{sid}, P', \mathsf{role})$ from U and from S_{id} with matching roles: mark the session active and notify Sim.
- **Transmission.** On $(\mathsf{Send}, \mathsf{sid}, m)$ from an active partner P:
  - ◇ if A_K and A_T are both uncorrupted (*adversarial-network mode*): leak (sid, m, P) to Sim, and on Sim's instruction $(\mathsf{Deliver}, \mathsf{sid}, m')$, output $(\mathsf{Received}, \mathsf{sid}, m')$ to the other partner. Sim may also instruct $\mathcal{F}_{\mathsf{US}}$ to drop the message.
  - ◇ if A_K or A_T is corrupted (*secure-channel mode*): leak $(\mathsf{sid}, |m|, P)$ to Sim, then output $(\mathsf{Received}, \mathsf{sid}, m)$ to the other partner.

Figure 9: Functionality $\mathcal{F}_{\mathsf{US}}$ (new).

Guarantee. If both attestors are uncorrupted, $\mathcal{A}$ sees every byte on the U–S_{id} wire and may modify, reorder, or drop messages at will. If either is corrupted, $\mathcal{F}_{\mathsf{US}}$ acts as $\mathcal{F}_{\mathsf{SC}}$ restricted to the U–S_{id} link: $\mathcal{A}$ learns only message lengths and cannot affect delivery.

C Security of Distributed-AEAD

This section gives the formal counterpart to the informal security discussion of Section 4.2. We provide game-based definitions of the two security requirements claimed for D-AEAD – splitting confidentiality into role-specific games IND-CPA-$\mathcal{P}_K$ and IND-CCA-$\mathcal{P}_T$ to match the asymmetric corruption model, and formalizing ciphertext integrity as a single game INT-CTXT against a malicious data party – and prove in Theorem C.6 that the construction of Section 4.1 satisfies all three under two assumptions on its underlying primitives: joint pseudorandomness of (GenEncMat, GenTagMat) and one-time tag unforgeability of (ComputeTag, VerifyTag).

For the specific instantiations of Section C.4, both assumptions reduce in a single step to standard properties of the underlying primitives – PRF security of AES (resp. the ChaCha20 core) and unforgeability of the GHASH (resp. Poly1305), so the security of D-AEAD rests on the same foundations as the AEAD schemes we are interested in this work. Note that the two confidentiality games are not symmetric: IND-CPA-$\mathcal{P}_K$ holds information-theoretically, because $\mathcal{P}_K$ is shielded from the plaintext at SR by a one-time pad applied by $\mathcal{P}_D$ before $\mathcal{P}_K$ enters the protocol, whereas IND-CCA-$\mathcal{P}_T$ is computational and reduces to the pseudorandomness of the keystream and tag-material primitives.

Notation. For a corrupt role $X \in \{K, C\}$, we write view_X for the view of $\mathcal{P}_X$ in a single honest execution of Π_{AEnc} or Π_{ADec}: that party's private input, internal randomness, and incoming messages on its channels; outgoing messages and internal computations are deterministic functions of these. We write $\overline{\mathsf{SR}} = [\ell] \setminus \mathsf{SR}$ for the public positions of an ℓ-bit plaintext. The confidentiality games below treat SR as part of the public common input; only the message contents at the positions indexed by SR are required to be hidden from $\mathcal{P}_K$ and $\mathcal{P}_T$.

We formalize below the two security requirements of Section 4.2. Confidentiality is split per honest-but-curious role into IND-CPA-$\mathcal{P}_K$ (Definition C.1) and IND-CCA-$\mathcal{P}_T$ (Definition C.2); ciphertext integrity against a malicious data party is captured by INT-CTXT (Definition C.3). The reduction to assumptions on the underlying primitives is stated in Theorem C.6.

C.1 Security Properties

Confidentiality against $\mathcal{P}_K$. A scheme D-AEAD provides confidentiality against $\mathcal{P}_K$ if no PPT adversary, given the long-term key k and the view view_K that $\mathcal{P}_K$ would obtain in an honest execution of Π_{AEnc} run on one of two adversarially chosen, equal-length plaintexts P_0, P_1 agreeing on $\overline{\mathsf{SR}}$, can determine which of the two plaintexts was the input to that execution. The mechanism that protects $P_b|_{\mathsf{SR}}$ from $\mathcal{P}_K$ is the one-time-pad masking $\widetilde{P} = P_b \oplus \mathsf{embed}_{\mathsf{SR},\ell}(M)$ that $\mathcal{P}_D$ applies before $\mathcal{P}_K$ enters the protocol. Accordingly, the bound on the IND-CPA-$\mathcal{P}_K$ advantage is information-theoretic and uses none of the assumptions on D-AEAD's underlying primitives. Restriction to a single challenge with no oracle access is without loss of generality: since $\mathcal{A}$ holds k and the common inputs are public, every honest action of $\mathcal{P}_K$ and $\mathcal{P}_T$ in Π_{AEnc} or Π_{ADec} is a deterministic function of public quantities together with k, so $\mathcal{A}$ may simulate any number of further encryption and decryption sessions locally. In Π_{ADec} specifically, $\mathcal{P}_K$'s incoming messages reduce to $|C|$ and the verdict bit v: the length $|C|$ leaks only the plaintext length, which is already pinned down by the equal-length constraint $|P_0| = |P_1|$, and the verdict $v = \mathsf{VerifyTag}(\mathsf{GenTagMat}(k, n), AD, C, T)$ is a deterministic function of k and the queried tuple and hence computable by $\mathcal{A}$. More formally:

Definition C.1 (Confidentiality against $\mathcal{P}_K$) A scheme D-AEAD provides confidentiality against $\mathcal{P}_K$ if for every PPT $\mathcal{A}$,

$$\begin{aligned} \mathsf{Adv}^{\text{IND-CPA-}\mathcal{P}_K}_{\text{D-AEAD},\mathcal{A}}(\lambda) &= \left|\Pr[\mathsf{Exp}^{\text{IND-CPA-}\mathcal{P}_K\text{-}1}_{\text{D-AEAD},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{IND-CPA-}\mathcal{P}_K\text{-}0}_{\text{D-AEAD},\mathcal{A}}(\lambda) = 1]\right| \\ &\leq \mathsf{negl}(\lambda), \end{aligned}$$

where $\mathsf{Exp}^{\text{IND-CPA-}\mathcal{P}_K\text{-}b}_{\text{D-AEAD},\mathcal{A}}$ is defined in figure 10.

Note that the view view_K of $\mathcal{P}_K$ in the honest Π_{AEnc} run on common input $(\mathsf{sid}, n, AD, \ell, \mathsf{SR})$ and private input k is reduced to the single incoming message $\widetilde{P}$: the remaining elements $K_s = \mathsf{GenEncMat}(k, n, \ell)$, $\boldsymbol{\tau} = \mathsf{GenTagMat}(k, n)$, and $\widetilde{C} = \widetilde{P} \oplus K_s$ are deterministic functions of $(k, n, \ell, \widetilde{P})$, already in $\mathcal{A}$'s state.

Confidentiality against $\mathcal{P}_T$. A scheme D-AEAD provides confidentiality against $\mathcal{P}_T$ if no PPT adversary, given the view view_C that $\mathcal{P}_T$ would obtain in a single challenge Π_{AEnc} execution on one of two adversarially chosen, equal-length plaintexts P_0, P_1 agreeing on $\overline{\mathsf{SR}}$, and adaptive oracle access to honest executions of Π_{AEnc} (the encryption oracle $\mathcal{O}_{\mathsf{Enc}}$, nonce-respecting, with the challenge nonce excluded from later queries) and Π_{ADec} (the decryption oracle $\mathcal{O}_{\mathsf{Dec}}$, with the challenge ciphertext-tag tuple excluded from later queries), can determine which of the two plaintexts was the input to the challenge execution made through the

Experiment $\mathsf{Exp}^{\mathsf{IND\text{-}CPA\text{-}}\mathcal{P}_K\text{-}b}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda)$:

1. $k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$;
2. $(\mathsf{sid}, n, AD, \ell, \mathsf{SR}, P_0, P_1, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda, k)$;
3. if $|P_0| \neq \ell$ or $|P_1| \neq \ell$ or $P_0|_{\overline{\mathsf{SR}}} \neq P_1|_{\overline{\mathsf{SR}}}$, then abort;
4. $M \xleftarrow{\$} \{0,1\}^{|\mathsf{SR}|}$; $\widetilde{P} \leftarrow P_b \oplus \mathsf{embed}_{\mathsf{SR},\ell}(M)$;
5. $b' \leftarrow \mathcal{A}(\mathsf{st}, \widetilde{P})$; return b'.

Figure 10: Experiment $\mathsf{IND\text{-}CPA\text{-}}\mathcal{P}_K$

challenge oracle $\mathcal{O}_{\mathsf{Chal}\text{-}b}$. Unlike in $\mathsf{IND\text{-}CPA\text{-}}\mathcal{P}_K$, the relevant protection here is the AEAD encryption step itself: $\mathcal{P}_T$ sees the ciphertext $C = P_b \oplus K_s$ but never sees the keystream K_s or the masked plaintext $\widetilde{P}$ from which K_s could be recovered, so the $\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T$ advantage is bounded computationally and the reduction goes through pseudorandomness of $(\mathsf{GenEncMat}, \mathsf{GenTagMat})$. More formally:

Definition C.2 (Confidentiality against $\mathcal{P}_T$) A scheme D-AEAD provides confidentiality against $\mathcal{P}_T$ if for every PPT $\mathcal{A}$,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda) &= \left|\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T\text{-}1}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T\text{-}0}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda) = 1]\right| \\ &\leq \mathsf{negl}(\lambda),\end{aligned}$$

where $\mathsf{Exp}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T\text{-}b}_{\mathsf{D\text{-}AEAD},\mathcal{A}}$ together with its oracles $\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Chal}\text{-}b}$ is defined in figure 11.

Experiment $\mathsf{Exp}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T\text{-}b}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda)$:

1. $k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$; $\mathcal{N}_{\mathsf{used}} \leftarrow \emptyset$; $\mathsf{ch} \leftarrow \bot$;
2. $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Chal}\text{-}b}}(1^\lambda)$;
3. return b'.

Oracle $\mathcal{O}_{\mathsf{Enc}}(n, AD, P, \mathsf{SR})$:

1. if $n \in \mathcal{N}_{\mathsf{used}}$ return $\bot$; otherwise $\mathcal{N}_{\mathsf{used}} \leftarrow \mathcal{N}_{\mathsf{used}} \cup \{n\}$;
2. $M \xleftarrow{\$} \{0,1\}^{|\mathsf{SR}|}$;
3. $\widetilde{P} \leftarrow P \oplus \mathsf{embed}_{\mathsf{SR},|P|}(M)$;
4. $K_s \leftarrow \mathsf{GenEncMat}(k, n, |P|)$;
5. $\boldsymbol{\tau} \leftarrow \mathsf{GenTagMat}(k, n)$;
6. $\widetilde{C} \leftarrow \widetilde{P} \oplus K_s$; $C \leftarrow \widetilde{C} \oplus \mathsf{embed}_{\mathsf{SR},|P|}(M)$;
7. $T \leftarrow \mathsf{ComputeTag}(\boldsymbol{\tau}, AD, C)$;
8. return $(M, \widetilde{C}, \boldsymbol{\tau}, C, T)$.

Oracle $\mathcal{O}_{\mathsf{Dec}}(n, AD, C, T)$:

1. if $(n, AD, C, T) = \mathsf{ch}$ return $\bot$;
2. $\boldsymbol{\tau} \leftarrow \mathsf{GenTagMat}(k, n)$;
3. $v \leftarrow \mathsf{VerifyTag}(\boldsymbol{\tau}, AD, C, T)$;
4. return $(\boldsymbol{\tau}, v)$.

Oracle $\mathcal{O}_{\mathsf{Chal}\text{-}b}(n^*, AD^*, \mathsf{SR}^*, P_0, P_1)$:

1. if $|P_0| \neq |P_1|$, or $P_0|_{\overline{\mathsf{SR}^*}} \neq P_1|_{\overline{\mathsf{SR}^*}}$, or $n^* \in \mathcal{N}_{\mathsf{used}}$, or $\mathsf{ch} \neq \bot$, return $\bot$;
2. $\mathcal{N}_{\mathsf{used}} \leftarrow \mathcal{N}_{\mathsf{used}} \cup \{n^*\}$;
3. compute $(M^*, \widetilde{C}^*, \boldsymbol{\tau}^*, C^*, T^*)$ by running the body of $\mathcal{O}_{\mathsf{Enc}}$ on $(n^*, AD^*, P_b, \mathsf{SR}^*)$, omitting its freshness check;
4. $\mathsf{ch} \leftarrow (n^*, AD^*, C^*, T^*)$;
5. return $(M^*, \widetilde{C}^*, \boldsymbol{\tau}^*, C^*, T^*)$.

Figure 11: Experiment $\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T$ and oracles $\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Chal}\text{-}b}$

Note that the returned tuples coincide with view_C in an honest run: in Π_{AEnc}, $\mathcal{P}_T$ receives M from $\mathcal{P}_D$ and $(\widetilde{C}, \boldsymbol{\tau})$ from $\mathcal{P}_K$ and emits (C, T) to $\mathcal{P}_D$; in Π_{ADec}, $\mathcal{P}_T$ receives (C, T) from $\mathcal{P}_D$ (already in the query) and $\boldsymbol{\tau}$ from $\mathcal{P}_K$, and emits the verdict v.

Ciphertext integrity. A scheme D-AEAD provides ciphertext integrity if no PPT adversary, given adaptive access to an encryption oracle $\mathcal{O}_{\mathsf{Enc}}$ (returning the ciphertext-tag pair an honest Π_{AEnc} run would produce on an adversarially chosen plaintext, nonce-respecting on the encryption side) and a decryption oracle $\mathcal{O}_{\mathsf{Dec}}$ (returning the verdict an honest Π_{ADec} run would produce on an adversarially chosen tuple, with no nonce

restriction), can submit to $\mathcal{O}_{\mathsf{Dec}}$ a tuple (n^*, AD^*, C^*, T^*) that was not previously delivered by $\mathcal{O}_{\mathsf{Enc}}$ and that Π_{ADec} accepts. The two oracles capture exactly the leverage available to a malicious $\mathcal{P}_D$ when interacting with honest $\mathcal{P}_K$ and $\mathcal{P}_T$: every outgoing message $\mathcal{P}_D$ can place on its Π_{AEnc} or Π_{ADec} channels induces the same honest-party output as a single $\mathcal{O}_{\mathsf{Enc}}$ or $\mathcal{O}_{\mathsf{Dec}}$ call, respectively; the formal protocol-to-game reduction is given inside the proof of Theorem C.6.

Definition C.3 (Ciphertext integrity) A scheme D-AEAD provides ciphertext integrity if for every PPT $\mathcal{A}$, the advantage

$$\mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda) \;=\; \Pr[\mathsf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda) = 1] \le \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD},\mathcal{A}}$ together with its oracles $\mathcal{O}_{\mathsf{Enc}}$ and $\mathcal{O}_{\mathsf{Dec}}$ is defined in figure 12.

Experiment $\mathsf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda)$:

1. $k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$; $\mathcal{N}_{\mathsf{used}}, \mathcal{Q} \leftarrow \emptyset$;
2. $\mathsf{win} \leftarrow 0$;
3. run $\mathcal{A}^{\mathcal{O}_{\mathsf{Enc}},\mathcal{O}_{\mathsf{Dec}}}(1^\lambda)$;
4. return win.

Oracle $\mathcal{O}_{\mathsf{Dec}}(n, AD, C, T)$:

1. $\boldsymbol{\tau} \leftarrow \mathsf{GenTagMat}(k, n)$;
2. $v \leftarrow \mathsf{VerifyTag}(\boldsymbol{\tau}, AD, C, T)$;
3. if $v = 1$ and $(n, AD, C, T) \notin \mathcal{Q}$, set $\mathsf{win} \leftarrow 1$;
4. return v.

Oracle $\mathcal{O}_{\mathsf{Enc}}(n, AD, P)$:

1. if $n \in \mathcal{N}_{\mathsf{used}}$ return $\perp$; otherwise $\mathcal{N}_{\mathsf{used}} \leftarrow \mathcal{N}_{\mathsf{used}} \cup \{n\}$;
2. $K_s \leftarrow \mathsf{GenEncMat}(k, n, |P|)$;
3. $C \leftarrow P \oplus K_s$;
4. $\boldsymbol{\tau} \leftarrow \mathsf{GenTagMat}(k, n)$;
5. $T \leftarrow \mathsf{ComputeTag}(\boldsymbol{\tau}, AD, C)$;
6. $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(n, AD, C, T)\}$;
7. return (C, T).

Figure 12: Experiment INT-CTXT and oracles $\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Dec}}$.

Note that the $\mathcal{O}_{\mathsf{Enc}}$ output (C, T) coincides with what a (possibly malicious) $\mathcal{P}_D$ receives from $\mathcal{P}_T$ at the end of an honest Π_{AEnc} run on the effective plaintext induced by $\mathcal{P}_D$'s outgoing messages; the $\mathcal{O}_{\mathsf{Dec}}$ verdict v coincides with the verdict $\mathcal{P}_D$ obtains from honest $\mathcal{P}_K$ and $\mathcal{P}_T$ at the end of Π_{ADec} on (n, AD, C, T).

C.2 Assumptions on the underlying primitives

The reduction in Theorem C.6 uses two assumptions on the underlying primitives: joint pseudorandomness of $(\mathsf{GenEncMat}, \mathsf{GenTagMat})$ and one-time tag unforgeability of $(\mathsf{ComputeTag}, \mathsf{VerifyTag})$. Both are properties of the components in isolation, and both reduce in a single step to standard primitive properties for the concrete schemes of Section C.4.

Joint pseudorandomness. A pair $(\mathsf{GenEncMat}, \mathsf{GenTagMat})$ is *jointly pseudorandom* if no PPT adversary distinguishes the pair of oracles $(\mathcal{O}_{\mathsf{S}}, \mathcal{O}_{\mathsf{T}})$ instantiated with a fresh key $k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$ from the same pair instantiated with uniformly sampled outputs. The stream oracle $\mathcal{O}_{\mathsf{S}}$ is nonce-respecting and returns a keystream of the requested length on each fresh nonce; the tag-material oracle $\mathcal{O}_{\mathsf{T}}$ accepts repeated queries on the same nonce and returns the same value on each repetition, capturing the fact that $\mathsf{GenTagMat}(k, n)$ depends only on (k, n) and is executed both at encryption and at every decryption attempt on n. Without loss of generality we assume the tag-material space of form $\mathcal{TM} = \mathcal{SP} \times \mathcal{TM}_\tau$ for some sets $\mathcal{SP}$ (the *shared subkey* space) and $\mathcal{TM}_\tau$ (the *per-nonce material* space). We require that $\mathsf{GenTagMat}$ decomposes as $\mathsf{GenTagMat}(k, n) = (\sigma(k), \tau(k, n))$ for some functions $\sigma : \mathcal{K} \to \mathcal{SP}$ and $\tau : \mathcal{K} \times \mathcal{N} \to \mathcal{TM}_\tau$, that is, the first coordinate of $\boldsymbol{\tau}^{(n)}$ depends only on k. When $\mathcal{SP} = \{\perp\}$ the decomposition is trivial. More formally:

Definition C.4 (Joint pseudorandomness) A pair $(\mathsf{GenEncMat}, \mathsf{GenTagMat})$ is *jointly pseudorandom* if for every PPT $\mathcal{A}$,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{jPR}}_{(\mathsf{GenEncMat},\mathsf{GenTagMat}),\mathcal{A}}(\lambda) &= \left|\Pr[\mathsf{Exp}^{\mathsf{jPR}\text{-}1}_{\mathcal{A}}(\lambda)=1] - \Pr[\mathsf{Exp}^{\mathsf{jPR}\text{-}0}_{\mathcal{A}}(\lambda)=1]\right| \\ &\leq \mathsf{negl}(\lambda),\end{aligned}$$

where $\mathsf{Exp}^{\mathsf{jPR}\text{-}b}_{\mathcal{A}}$ is defined in figure 13.

Experiment $\mathsf{Exp}^{\mathsf{jPR}\text{-}b}_{(\mathsf{GenEncMat},\mathsf{GenTagMat}),\mathcal{A}}(\lambda)$:

1. $\mathcal{N}^S_{\mathsf{used}} \leftarrow \emptyset$; $\mathsf{T} \leftarrow [\,]$;
2. if $b=1$: $k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$;
3. if $b=0$: $\sigma \xleftarrow{\$} \mathcal{SP}$;
4. $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{S}},\mathcal{O}_{\mathsf{T}}}(1^\lambda)$; return b'.

Oracle $\mathcal{O}_{\mathsf{S}}(n,\ell)$:

1. if $n \in \mathcal{N}^S_{\mathsf{used}}$ return $\perp$; otherwise $\mathcal{N}^S_{\mathsf{used}} \leftarrow \mathcal{N}^S_{\mathsf{used}} \cup \{n\}$;
2. if $b=1$ return $\mathsf{GenEncMat}(k,n,\ell)$ else return $K_s \xleftarrow{\$} \{0,1\}^\ell$.

Oracle $\mathcal{O}_{\mathsf{T}}(n)$:

1. if $\mathsf{T}[n] \neq \perp$ return $\mathsf{T}[n]$;
2. if $b=1$ set $\mathsf{T}[n] \leftarrow \mathsf{GenTagMat}(k,n)$;
3. else: $\tau_n \xleftarrow{\$} \mathcal{TM}_\tau$, set $\mathsf{T}[n] \leftarrow (\sigma,\tau_n)$;
4. return $\mathsf{T}[n]$.

Figure 13: Experiment jPR over the product structure $\mathcal{TM} = \mathcal{SP} \times \mathcal{TM}_\tau$.

One-time tag unforgeability. Sample a shared subkey $\sigma \xleftarrow{\$} \mathcal{SP}$ once and, on the first appearance of each nonce n, a per-nonce material $\tau_n \xleftarrow{\$} \mathcal{TM}_\tau$. The adversary is allowed at most one $\mathcal{O}_{\mathsf{Tag}}$ call per nonce and arbitrarily many $\mathcal{O}_{\mathsf{Ver}}$ calls on arbitrary nonces; no PPT adversary can produce a fresh tuple (n, AD, C, T) that VerifyTag accepts. More formally:

Definition C.5 (One-time tag unforgeability) A pair $(\mathsf{ComputeTag}, \mathsf{VerifyTag})$ is *one-time tag unforgeable* if for every PPT $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{OTUF}}_{(\mathsf{ComputeTag},\mathsf{VerifyTag}),\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{\mathsf{OTUF}}_{\mathcal{A}}(\lambda)=1] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{OTUF}}_{\mathcal{A}}$ and its oracles $\mathcal{O}_{\mathsf{Tag}}, \mathcal{O}_{\mathsf{Ver}}$ are defined in figure 14.

Experiment $\mathsf{Exp}^{\mathsf{OTUF}}_{(\mathsf{ComputeTag},\mathsf{VerifyTag}),\mathcal{A}}(\lambda)$:

1. $\sigma \xleftarrow{\$} \mathcal{SP}$; $\mathsf{T} \leftarrow [\,]$; $\mathcal{S} \leftarrow \emptyset$; $\mathsf{Tagged} \leftarrow \emptyset$; $\mathsf{win} \leftarrow 0$;
2. run $\mathcal{A}^{\mathcal{O}_{\mathsf{Tag}},\mathcal{O}_{\mathsf{Ver}}}(1^\lambda)$; return win.

Oracle $\mathcal{O}_{\mathsf{Tag}}(n, AD, C)$:

1. if $n \in \mathsf{Tagged}$ return $\perp$; otherwise $\mathsf{Tagged} \leftarrow \mathsf{Tagged} \cup \{n\}$;
2. if $\mathsf{T}[n] = \perp$: $\tau_n \xleftarrow{\$} \mathcal{TM}_\tau$, $\mathsf{T}[n] \leftarrow (\sigma,\tau_n)$;
3. $T \leftarrow \mathsf{ComputeTag}(\mathsf{T}[n], AD, C)$; $\mathcal{S} \leftarrow \mathcal{S} \cup \{(n, AD, C, T)\}$; return T.

Oracle $\mathcal{O}_{\mathsf{Ver}}(n, AD, C, T)$:

1. if $\mathsf{T}[n] = \perp$: $\tau_n \xleftarrow{\$} \mathcal{TM}_\tau$, $\mathsf{T}[n] \leftarrow (\sigma,\tau_n)$;
2. $v \leftarrow \mathsf{VerifyTag}(\mathsf{T}[n], AD, C, T)$;
3. if $v=1$ and $(n, AD, C, T) \notin \mathcal{S}$, set $\mathsf{win} \leftarrow 1$; return v.

Figure 14: Experiment OTUF with shared subkey σ and per-nonce one-time tag access.

C.3 Main theorem

Theorem C.6 (Security of D-AEAD) Let $\mathcal{A}$ be a PPT adversary making at most q_e queries to $\mathcal{O}_{\mathsf{Enc}}$ and q_d queries to $\mathcal{O}_{\mathsf{Dec}}$ in the experiments where these oracles are present.

1. $\mathsf{Adv}^{\mathsf{IND\text{-}CPA\text{-}}\mathcal{P}_K}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda) = 0$.
2. There exists a PPT adversary $\mathcal{B}$ such that

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda) \le 2 \cdot \mathsf{Adv}^{\mathsf{jPR}}_{(\mathsf{GenEncMat},\mathsf{GenTagMat}),\mathcal{B}}(\lambda).$$

3. There exist PPT adversaries $\mathcal{B}, \mathcal{C}$ such that

$$\mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\mathsf{jPR}}_{(\mathsf{GenEncMat},\mathsf{GenTagMat}),\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{OTUF}}_{(\mathsf{ComputeTag},\mathsf{VerifyTag}),\mathcal{C}}(\lambda).$$

Proof. We treat the three claims in turn.

Confidentiality against $\mathcal{P}_K$. The bit b enters $\mathsf{Exp}^{\mathsf{IND\text{-}CPA\text{-}}\mathcal{P}_K\text{-}b}$ only through $\widetilde{P} = P_b \oplus \mathsf{embed}_{\mathsf{SR},\ell}(M)$, where $M \xleftarrow{\$} \{0,1\}^{|\mathsf{SR}|}$ is uniform and independent of P_b. At positions in SR, $\widetilde{P}|_{\mathsf{SR}} = P_b|_{\mathsf{SR}} \oplus M$ is a one-time-pad encryption of $P_b|_{\mathsf{SR}}$ under the uniform key M, so $\widetilde{P}|_{\mathsf{SR}}$ is uniform on $\{0,1\}^{|\mathsf{SR}|}$ irrespective of b. At positions in $\overline{\mathsf{SR}}$ the embedded mask is zero, so $\widetilde{P}|_{\overline{\mathsf{SR}}} = P_b|_{\overline{\mathsf{SR}}}$, which equals $P_0|_{\overline{\mathsf{SR}}} = P_1|_{\overline{\mathsf{SR}}}$. The distribution of $\widetilde{P}$ is therefore independent of b, and the advantage is zero.

Confidentiality against $\mathcal{P}_T$. Define the hybrid experiment Hyb^b as $\mathsf{Exp}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T\text{-}b}$ with the following modification: on first occurrence of nonce n, sample $\boldsymbol{\tau}^{(n)} \xleftarrow{\$} \mathcal{TM}$ and cache it; whenever $\mathcal{O}_{\mathsf{Enc}}$ or $\mathcal{O}_{\mathsf{Chal}\text{-}b}$ first uses n, additionally sample $K_s^{(n)} \xleftarrow{\$} \{0,1\}^{|P|}$ and cache it. All subsequent invocations of $\mathsf{GenEncMat}(k,n,\cdot)$ or $\mathsf{GenTagMat}(k,n)$ on n return the cached values. We bound the advantage in two steps.

Step 1: independence of the hybrid from b. The bit b enters Hyb^b only through the $\mathcal{O}_{\mathsf{Chal}\text{-}b}$ call on P_b. Inside $\mathcal{O}_{\mathsf{Chal}\text{-}b}$, the keystream K_s^* is uniform on $\{0,1\}^{|P_b|}$ and independent of M^* and $\boldsymbol{\tau}^*$, so $C^* = P_b \oplus K_s^*$ is uniform on $\{0,1\}^{|P_b|}$ and independent of $(M^*, \boldsymbol{\tau}^*)$. The remaining components of the challenge tuple are deterministic functions of $(M^*, \boldsymbol{\tau}^*, C^*)$ and of $AD^*, \mathsf{SR}^*, |P_b|$: $\widetilde{C}^* = C^* \oplus \mathsf{embed}_{\mathsf{SR}^*,|P_b|}(M^*)$ and $T^* = \mathsf{ComputeTag}(\boldsymbol{\tau}^*, AD^*, C^*)$. Since $\mathcal{O}_{\mathsf{Chal}\text{-}b}$ enforces $|P_0| = |P_1|$, the joint distribution of $(M^*, \widetilde{C}^*, \boldsymbol{\tau}^*, C^*, T^*)$ is identical for $b = 0$ and $b = 1$. The remaining oracles take inputs from $\mathcal{A}$ that are not functions of b and produce outputs by procedures that do not branch on b. Therefore the joint distribution of $\mathcal{A}$'s view in Hyb^b is identical for $b \in \{0,1\}$, so $\Pr[\mathsf{Hyb}^0 = 1] = \Pr[\mathsf{Hyb}^1 = 1]$.

Step 2: reduction to jPR. Define a PPT adversary $\mathcal{B}$ against jPR as follows.

1. Receive 1^λ and oracle access to $(\mathcal{O}_{\mathsf{S}}, \mathcal{O}_{\mathsf{T}})$ of the jPR experiment, and sample $c \xleftarrow{\$} \{0,1\}$.
2. Run $\mathcal{A}$ and simulate $\mathsf{Exp}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T\text{-}c}_{\mathsf{D\text{-}AEAD},\mathcal{A}}$, with each invocation of $\mathsf{GenEncMat}(k,n,\ell)$ inside $\mathcal{O}_{\mathsf{Enc}}$ or $\mathcal{O}_{\mathsf{Chal}\text{-}b}$ replaced by a call to $\mathcal{O}_{\mathsf{S}}(n,\ell)$ and each invocation of $\mathsf{GenTagMat}(k,n)$ inside $\mathcal{O}_{\mathsf{Enc}}$, $\mathcal{O}_{\mathsf{Dec}}$, or $\mathcal{O}_{\mathsf{Chal}\text{-}b}$ replaced by a call to $\mathcal{O}_{\mathsf{T}}(n)$.
3. When $\mathcal{A}$ halts and outputs a bit b', return $[b' = c]$.

When $(\mathcal{O}_{\mathsf{S}}, \mathcal{O}_{\mathsf{T}})$ are the real jPR oracles, $\mathcal{B}$'s simulation is distributed exactly as $\mathsf{Exp}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T\text{-}c}_{\mathsf{D\text{-}AEAD},\mathcal{A}}$, so $\Pr[\mathcal{B}{=}1 \mid \mathsf{jPR}\text{-}1] = \frac{1}{2} + \frac{1}{2}\big(\Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T\text{-}1}_{\mathsf{D\text{-}AEAD},\mathcal{A}}{=}1] - \Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T\text{-}0}_{\mathsf{D\text{-}AEAD},\mathcal{A}}{=}1]\big)$. When $(\mathcal{O}_{\mathsf{S}}, \mathcal{O}_{\mathsf{T}})$ are the ideal jPR oracles, the simulation is distributed exactly as Hyb^c, and by Step 1 the output of $\mathcal{A}$ in Hyb^c is independent of c, so $\Pr[\mathcal{B}{=}1 \mid \mathsf{jPR}\text{-}0] = \frac{1}{2}$. Taking the absolute difference, $\mathsf{Adv}^{\mathsf{jPR}}_{(\mathsf{GenEncMat},\mathsf{GenTagMat}),\mathcal{B}}(\lambda) = \frac{1}{2}\,\mathsf{Adv}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda)$, which gives the stated bound $\mathsf{Adv}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda) \le 2 \cdot \mathsf{Adv}^{\mathsf{jPR}}_{(\mathsf{GenEncMat},\mathsf{GenTagMat}),\mathcal{B}}(\lambda)$.

Ciphertext integrity. Define the hybrid experiment Hyb as $\mathsf{Exp}^{\mathsf{INT\text{-}CTXT}}$ with the same modification as above: each $(K_s^{(n)}, \boldsymbol{\tau}^{(n)})$ is sampled uniformly on first use of n and cached. We bound the advantage in two steps.

Step 1: indistinguishability from the hybrid. Define a PPT adversary $\mathcal{B}$ against jPR by the same construction as in the $\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T$ proof, with the simulated experiment being $\mathsf{Exp}^{\mathsf{INT\text{-}CTXT}}$ and the returned value being the experiment's win bit. By the same argument, $|\Pr[\mathsf{Exp}^{\mathsf{INT\text{-}CTXT}} = 1] - \Pr[\mathsf{Hyb} = 1]| = \mathsf{Adv}^{\mathsf{jPR}}_{\mathcal{B}}(\lambda)$.

Step 2: bounding the hybrid via OTUF. Define a PPT adversary $\mathcal{C}$ against OTUF as follows.

1. Receive 1^λ and oracle access to $(\mathcal{O}_{\mathsf{Tag}}, \mathcal{O}_{\mathsf{Ver}})$ of the OTUF experiment; initialize $\mathcal{N}_{\mathsf{used}}, \mathcal{Q} \leftarrow \emptyset$.
2. Run $\mathcal{A}$ and simulate Hyb as follows.
3. On every $\mathcal{O}_{\mathsf{Enc}}(n, AD, P)$ call: if $n \in \mathcal{N}_{\mathsf{used}}$ return $\perp$, otherwise $\mathcal{N}_{\mathsf{used}} \leftarrow \mathcal{N}_{\mathsf{used}} \cup \{n\}$; sample $K_s \xleftarrow{\$} \{0,1\}^{|P|}$; set $C \leftarrow P \oplus K_s$; obtain $T \leftarrow \mathcal{O}_{\mathsf{Tag}}(n, AD, C)$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(n, AD, C, T)\}$; return (C, T).
4. On every $\mathcal{O}_{\mathsf{Dec}}(n, AD, C, T)$ call: obtain $v \leftarrow \mathcal{O}_{\mathsf{Ver}}(n, AD, C, T)$ and return v.
5. Halt when $\mathcal{A}$ halts.

The shared subkey σ and each per-nonce τ_n in $\mathcal{C}$'s simulation are sampled by the OTUF experiment exactly as Hyb would sample them, and each K_s is uniform on $\{0,1\}^{|P|}$, so $\mathcal{A}$'s view under $\mathcal{C}$ is distributed exactly as in Hyb. The set $\mathcal{Q}$ maintained by $\mathcal{C}$ coincides with the OTUF experiment's $\mathcal{S}$, so $\mathsf{win} = 1$ in OTUF exactly when $\mathcal{A}$'s $\mathcal{O}_{\mathsf{Dec}}$ query returns 1 on a tuple absent from $\mathcal{Q}$, which is the win condition of Hyb. Hence $\mathsf{Adv}^{\mathsf{OTUF}}_{(\mathsf{ComputeTag},\mathsf{VerifyTag}),\mathcal{C}}(\lambda) = \Pr[\mathsf{Hyb} = 1]$.

Combining. $\mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD},\mathcal{A}}(\lambda) \le |\Pr[\mathsf{Exp}^{\mathsf{INT\text{-}CTXT}} = 1] - \Pr[\mathsf{Hyb} = 1]| + \Pr[\mathsf{Hyb} = 1] = \mathsf{Adv}^{\mathsf{jPR}}_{(\mathsf{GenEncMat},\mathsf{GenTagMat}),\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{OTUF}}_{(\mathsf{ComputeTag},\mathsf{VerifyTag}),\mathcal{C}}(\lambda)$.

C.4 Concrete schemes

AES-GCM [Dwo07]. Let $\kappa \in \{128, 256\}$. Set $|\mathcal{K}| = \kappa$, $|\mathcal{N}| = 96$, $|\mathcal{T}| = 128$, $\mathcal{SP} = \{0,1\}^{128}$, $\mathcal{TM}_\tau = \{0,1\}^{128}$, so $\mathcal{TM} = \mathcal{SP} \times \mathcal{TM}_\tau$. Let $\mathsf{AES}_k : \{0,1\}^{128} \to \{0,1\}^{128}$ denote AES block encryption with κ-bit key, $\mathsf{CTR}_k(n, \ell)$ the AES-CTR keystream of length ℓ with 32-bit block counter starting at 2 per [Dwo07], and GHASH_H the polynomial-evaluation hash over $\mathrm{GF}(2^{128})$ with subkey H, applied to the standard GCM padding of (AD, C). Define

$$\begin{aligned}
\mathsf{GenEncMat}(k, n, \ell) &= \mathsf{CTR}_k(n, \ell), \\
\mathsf{GenTagMat}(k, n) &= (H, S), \quad H = \mathsf{AES}_k(0^{128}),\ S = \mathsf{AES}_k(n \,\|\, 0^{31} \,\|\, 1), \\
\mathsf{ComputeTag}\big((H, S), AD, C\big) &= \mathsf{GHASH}_H(AD, C) \oplus S, \\
\mathsf{VerifyTag}\big((H, S), AD, C, T\big) &= [\mathsf{GHASH}_H(AD, C) \oplus S = T],
\end{aligned}$$

where H depends only on k and instantiates the shared subkey $\sigma \in \mathcal{SP}$, while S instantiates the per-nonce material $\tau_n \in \mathcal{TM}_\tau$.
All AES calls land on pairwise disjoint inputs: 0^{128} for H, $n\|0^{31}\|1$ for S, and $n\|0^{31}\|c$ with $c \ge 2$ for $\mathsf{CTR}_k(n, \cdot)$. PRP/PRF switching replaces AES_k by a uniformly random function, yielding a single uniform H shared across all nonces, an independent uniform $S^{(n)}$ per nonce, and uniform keystreams – the distribution of the jPR ideal experiment of figure 13 – so jPR reduces in a single step to PRP security of AES_κ. With $(\sigma, \tau_n) = (H, S_n)$ both uniform on $\{0,1\}^{128}$ in OTUF, the ε-almost-XOR-universality of GHASH [MV04] bounds the forgery probability by $q_v \cdot \lceil \ell_{\max}/128 \rceil \cdot 2^{-128}$ over q_v verify queries on inputs of total length at most $\ell_{\max}$ bits, discharging OTUF unconditionally.

ChaCha20-Poly1305 [NL15]. Set $|\mathcal{K}| = 256$, $|\mathcal{N}| = 96$, $|\mathcal{T}| = 128$, $\mathcal{SP} = \{\perp\}$, $\mathcal{TM}_\tau = \{0,1\}^{256}$, so $\mathcal{TM} \cong \mathcal{TM}_\tau$ and the shared component is trivial. Let $\mathsf{ChaCha20}_k(n, c, \ell)$ denote the ChaCha20 keystream of ℓ bits under key k and 96-bit nonce n starting at 32-bit block counter c, and $\mathsf{Poly1305}_{(r,s)}$ the Poly1305 MAC under one-time key (r, s), applied to the standard RFC 8439 [NL15] padding of (AD, C). Define

$$\begin{aligned}
\mathsf{GenEncMat}(k, n, \ell) &= \mathsf{ChaCha20}_k(n, 1, \ell), \\
\mathsf{GenTagMat}(k, n) &= (r, s), \quad r \,\|\, s = \mathsf{ChaCha20}_k(n, 0, 256), \\
\mathsf{ComputeTag}\big((r, s), AD, C\big) &= \mathsf{Poly1305}_{(r,s)}(AD \,\|\, C), \\
\mathsf{VerifyTag}\big((r, s), AD, C, T\big) &= [\mathsf{Poly1305}_{(r,s)}(AD \,\|\, C) = T].
\end{aligned}$$

The counter-zero block consumed by $\mathsf{GenTagMat}$ and the counter-≥ 1 blocks consumed by $\mathsf{GenEncMat}$ are produced by disjoint inputs to the ChaCha20 core, so PRF security of the core reduces jPR in a single

step to a standard primitive assumption; with $\mathcal{SP}$ trivial there is no shared component to align. With each $\tau_n = (r_n, s_n) \xleftarrow{\$} \{0,1\}^{256}$ on first use of n in OTUF, the ε-almost-Δ-universality of the Poly1305 polynomial hash [Ber05] bounds the forgery probability by $q_v \cdot 8\lceil \ell_{\max}/16 \rceil \cdot 2^{-106}$ over q_v verify queries on inputs of total length at most $\ell_{\max}$ bytes, discharging OTUF unconditionally.

D Reclaim Protocol Security Proof

Theorem 5.1 (restated) Let D-AEAD be a distributed stream-cipher AEAD scheme satisfying the confidentiality and ciphertext-integrity guarantees of Section 4.2, let every $\mathsf{Prot}_{\mathsf{id}}$ in the deployment be request–reply and request-extractable (Definitions B.1 and B.2), and let every predicate be supported for its protocol: admissible (Definition B.5) and of the form fixed in Section 5.2. Then the Reclaim protocol $\mathcal{RP}$ of Section 5 UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{zkPoPE}}$ of Section 3 in the $(\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{KE}}, \mathcal{F}_{\mathsf{CERT}}, \mathcal{F}_{\mathsf{US}})$-hybrid model, under the admissible corruption pattern recalled in Section D.1.

D.1 Model and conventions

The functionality $\mathcal{F}_{\mathsf{US}}$ and the admissible corruption pattern were motivated in Subsection 2.2.1 of the Technical Overview. We restate them here for self-containment.

The User–Service channel $\mathcal{F}_{\mathsf{US}}$. $\mathcal{F}_{\mathsf{US}}$ is parameterized by the corruption status of the attestors. When A_K and A_T are both uncorrupted, $\mathcal{F}_{\mathsf{US}}$ behaves as a standard adversarial network: every wire byte is leaked to the adversary, and the adversary may modify, reorder or drop messages. When either attestor is corrupted, $\mathcal{F}_{\mathsf{US}}$ behaves as a UC secure channel: it leaks only message lengths and delivers every message unaltered.

Admissible corruption pattern. A static corruption choice (FC, HCC) is *admissible* if $FC \subseteq \{\mathsf{U}\}$, $HCC \subseteq \{\mathsf{A}_K, \mathsf{A}_T\}$, $FC \cap HCC = \emptyset$, $|HCC| \leq 1$, and $FC \neq \emptyset \implies HCC = \emptyset$. Services are honest by default; the Verifier is omitted from FC because proofs are treated as adversarial-visible by default and a corrupted $\mathcal{V}$ adds nothing beyond what the environment already has. The pattern reduces to four cases:

- **Case A (passive):** $FC = HCC = \emptyset$.
- **Case B (malicious U):** $FC = \{\mathsf{U}\}$, $HCC = \emptyset$.
- **Case C (HCC A_K):** $FC = \emptyset$, $HCC = \{\mathsf{A}_K\}$.
- **Case D (HCC A_T):** $FC = \emptyset$, $HCC = \{\mathsf{A}_T\}$.

$\mathcal{F}_{\mathsf{US}}$ is in adversarial-network mode in Cases A and B and in secure-channel mode in Cases C and D.

Mapping to D-AEAD's roles. Phases B and C of $\mathcal{RP}$ instantiate D-AEAD's sub-protocols Π_{AEnc} and Π_{ADec} with the role correspondence

$$\mathsf{U} \longleftrightarrow \mathcal{P}_D, \qquad \mathsf{A}_K \longleftrightarrow \mathcal{P}_K, \qquad \mathsf{A}_T \longleftrightarrow \mathcal{P}_T,$$

and the secret-index sets $\mathsf{SR}_{\mathsf{req}}$, $\mathsf{SR}_{\mathsf{resp}}$ on the request and response side. Under the admissible pattern, at most one of $\{\mathsf{U}, \mathsf{A}_K, \mathsf{A}_T\}$ is corrupted in any single execution, which matches D-AEAD's single-party threat model: $\mathsf{U} \in FC$ corresponds to D-AEAD's malicious-$\mathcal{P}_D$ regime, while $\mathsf{A}_K \in HCC$ or $\mathsf{A}_T \in HCC$ corresponds to its semi-honest $\mathcal{P}_K$ or $\mathcal{P}_T$ regime.

Conventions. We treat KDF as ideal: the tuple $(k_{\mathsf{enc}}, k_{\mathsf{dec}}, IV_{\mathsf{enc}}, IV_{\mathsf{dec}})$ derived from the uniform $\mathcal{F}_{\mathsf{KE}}$ key is uniform with independent components, so the two directions of a session are keyed independently. A pseudorandom KDF adds one standard term to every bound below. By the session-state convention of Section 5.3, every honest party invokes $(\mathsf{GenEncMat}, \mathsf{GenTagMat})$ at most once per direction per session, so each nonce $IV_{\mathsf{enc}}, IV_{\mathsf{dec}}$ is used once. A semi-honest attestor is uncorrupted for $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{F}_{\mathsf{CERT}}$ and $\mathcal{F}_{\mathsf{KE}}$ (Appendix B.3): its signatures are verified against the sign table of its $\mathcal{F}_{\mathsf{CERT}}$ instance, and a session key it shares is sampled by $\mathcal{F}_{\mathsf{KE}}$.
Throughout, q denotes the number of Prove sessions $\mathcal{Z}$ instantiates, and every D-AEAD advantage is that of an adversary running in time comparable to $\mathcal{Z}$ and $\mathcal{A}$ together.

D.2 Simulator

We construct a simulator Sim such that $\mathrm{REAL}_{\mathcal{RP},\mathcal{A},\mathcal{Z}} \approx_c \mathrm{IDEAL}_{\mathcal{F}_{\mathsf{zkPoPE}},\mathsf{Sim},\mathcal{Z}}$ for every admissible corruption pattern. Sim runs the four hybrid functionalities of Appendix B.3 internally and, for every uncorrupted party of $\mathcal{RP}$, an in-head copy of that party; a corrupted U is $\mathcal{A}$'s, and a semi-honest attestor is run by Sim with its view forwarded to $\mathcal{A}$. Sim draws every value it samples from its random tape ρ, one slice per session, so that a session's values are available wherever Sim needs them.

Setup. On $\mathcal{F}_{\mathsf{zkPoPE}}.\mathtt{Setup}$, Sim returns the empty string as crs and $\mathsf{Algs} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$, with Setup and Prove the corresponding protocols of Section 5.3 and $\mathsf{Verify}(\mathsf{crs}, \pi)$ the algorithm that parses π and checks σ_K and σ_T, over the components of π as in $\mathcal{RP}$'s Verify, under the Vfy algorithms Sim commits to $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_K]$ and $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_T]$. At the first activation of each of these two instances Sim supplies a correct $(\mathsf{Sign}, \mathsf{Vfy})$ pair; every π it later returns to $\mathcal{F}_{\mathsf{zkPoPE}}$ carries signatures produced by them, so the check at step 9 of $\mathtt{Prove}$ passes. Sim runs $\mathcal{F}_{\mathsf{CERT}}[S_{\mathsf{id}}]$ on behalf of the honest service, and configures $\mathcal{F}_{\mathsf{US}}$ to adversarial-network mode when $HCC = \emptyset$ and to secure-channel mode otherwise.

What Sim knows about a session. For a supported predicate Φ write $(\mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}})$ for its position sets, ℓ_{Req}, ℓ_{Resp} for the lengths its format fixes, and $Req^\star \in \{0,1\}^{\ell_{Req}}$, $Resp^\star \in \{0,1\}^{\ell_{Resp}}$ for the strings that carry, outside $\mathsf{SR}_{\mathsf{req}}$ and $\mathsf{SR}_{\mathsf{resp}}$, the byte values every pair satisfying $\widehat{\Phi}$ shares (Section 5.2), and zero on the two sets. All of these are functions of $(\Phi, \mathsf{Prot}_{\mathsf{id}})$. In the honest-User cases Sim learns $(P, \Phi, \mathsf{id}, res, \ell)$ from $\mathcal{F}_{\mathsf{zkPoPE}}$'s announcement, with ℓ the length of the reply, and nothing else about the session. When $res = 1$ the disclosed bytes of the real request and response are those of $(Req^\star, Resp^\star)$ and $\ell = \ell_{Resp}$. When $res = 0$ the real request still agrees with $Req^\star$ outside $\mathsf{SR}_{\mathsf{req}}$, by the check in step 2 of $\mathcal{RP}$'s Prove, while the response is unconstrained; we then let $Resp^\star = 0^\ell$, and the session ends before anything derived from the response other than its encryption appears in any view. In Case B, Sim reads the request from $\mathcal{A}$'s traffic and receives the response from $\mathcal{F}_{\mathsf{zkPoPE}}$.

Delayed outputs. Sim releases each pending output of $\mathcal{F}_{\mathsf{zkPoPE}}$ at the moment the corresponding party of $\mathcal{RP}$ produces it: $(\mathtt{Executed}, \mathsf{out}_{\mathsf{Service}})$ to the service when the in-head S_{id} accepts the request record; $(\mathtt{Executed}, \mathsf{reply})$ and, on $res = 0$, $(\mathtt{Prove}, \bot)$ to the caller when the in-head U completes Phase C; the proof when Phase D completes. A pending output to a party for which no session was announced is the $\bot$ of an input no reply can satisfy, and is released at once. A session that $\mathcal{A}$ stalls leaves its outputs undelivered, as in REAL. When the in-head A_T rejects the response record of an honest-User session, the honest U of $\mathcal{RP}$ outputs $\bot$; Sim reproduces this by delivering $\mathcal{F}_{\mathsf{zkPoPE}}$'s $\bot$ if $res = 0$, and by answering the proof request with $\bot$, which fails Verify, if $res = 1$.

Honest-User sessions (Cases A, C, D). $\mathcal{F}_{\mathsf{zkPoPE}}$'s announcement $(\mathsf{sid}, \mathtt{Prove}, P, \Phi, \mathsf{id}, res, \ell)$ is what tells Sim that a session of P for (Φ, id) has begun. Sim then runs $\mathcal{RP}$ with its in-head parties, letting the in-head U run on the request $Req^\star$ and the in-head S_{id} answer with the response $Resp^\star$ instead of evaluating $\mathsf{Prot}_{\mathsf{id}}$, while $\mathcal{A}$ acts on $\mathcal{F}_{\mathsf{US}}$ as its mode permits and receives the view of the semi-honest attestor, if any. Concretely, with wire messages delivered as $\mathcal{A}$ chooses in Case A and unaltered in Cases C and D:

- *Phase A.* The in-head A_K samples r, forms $\mathsf{sid}_{\mathsf{KE}}$ and $\mathsf{ssid} = (\mathsf{sid}, P, \mathsf{id}, r)$, and the handshake runs as specified over $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{F}_{\mathsf{US}}$, $\mathcal{F}_{\mathsf{CERT}}[S_{\mathsf{id}}]$ and $\mathcal{F}_{\mathsf{KE}}$; Sim's $\mathcal{F}_{\mathsf{KE}}$ hands both endpoints a κ drawn from ρ, from which they derive $(k_{\mathsf{enc}}, k_{\mathsf{dec}}, IV_{\mathsf{enc}}, IV_{\mathsf{dec}})$.
- *Phase B.* The in-head U samples M_{req} and sends $\widetilde{Req} = Req^\star \oplus \mathsf{embed}_{\mathsf{SR}_{\mathsf{req}},\ell_{Req}}(M_{\mathsf{req}})$ and $\mathsf{SR}_{\mathsf{req}}$ to A_K, and $\mathsf{embed}_{\mathsf{SR}_{\mathsf{req}},\ell_{Req}}(M_{\mathsf{req}})$ to A_T; the in-head attestors run Π_{AEnc} honestly, A_K forwarding $\mathsf{SR}_{\mathsf{req}}$, and U forwards $(Req_{\mathsf{enc}}, T_{\mathsf{req}})$ over $\mathcal{F}_{\mathsf{US}}$.
- *Phase C.* If the delivered request record is the one the in-head A_T produced, the in-head S_{id} encrypts $Resp^\star$ under $(k_{\mathsf{dec}}, IV_{\mathsf{dec}})$ and sends the record over $\mathcal{F}_{\mathsf{US}}$; otherwise it rejects. The in-head U and attestors run Π_{ADec} on the record $\mathcal{A}$ delivers, A_T accepting it only if it is the one S_{id} sent. On completion the in-head U halts if $res = 0$.

– *Phase D.* If $res = 1$, the in-head parties run Phase D as specified, and U assembles

$$\pi = (\sigma_K, \sigma_T, \mathsf{id}, \mathsf{ssid}, \mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}}, Req_{\mathsf{null}}, \widetilde{K}_{\mathsf{resp}}, \widetilde{Resp}_{\mathsf{enc}}).$$

Sim answers the proof request with this π as soon as the in-head A_T has issued σ_T, when every component of π is determined. In Case C it forwards to $\mathcal{A}$ the in-head A_K's view: its handshake state, the received $\widetilde{Req}$ and $\mathsf{SR}_{\mathsf{req}}$ and the material it computes in Phases B and C, and its Phase-D inputs and outputs. In Case D it forwards the in-head A_T's view: the mask received from U, the messages received from A_K, the records and verdicts it computes, and its Phase-D outputs. The simulation differs from a real execution in two respects: the in-head S_{id} and A_T accept only records their honest counterpart produced, which Sim, running both, can check; and the plaintexts $(Req, Resp)$ have been replaced by $(Req^\star, Resp^\star)$, which agree with them outside $(\mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}})$ whenever $res = 1$.

Malicious-User sessions (Case B). $\mathcal{A}$ drives U's side of every channel, and Sim's in-head A_K, A_T and S_{id} respond as $\mathcal{RP}$ prescribes, with one addition: the in-head S_{id} accepts a request record only if the in-head A_T produced it in that session, and the in-head A_T a response record only if the in-head S_{id} did, which Sim, running both, can check. $\mathcal{F}_{\mathsf{CERT}}[S_{\mathsf{id}}]$ accepts only pairs in its sign table, so a handshake context that $\mathcal{A}$ alters is rejected. The in-head S_{id} cannot evaluate $\mathsf{Prot}_{\mathsf{id}}$, holding neither $\mathsf{sec}_{\mathsf{id}}$ nor the service's coins, so Sim obtains the response from $\mathcal{F}_{\mathsf{zkPoPE}}$. It does so at the moment the in-head S_{id} accepts a request record $(Req_{\mathsf{enc}}, T_{\mathsf{req}})$ delivered over $\mathcal{F}_{\mathsf{US}}$, the moment at which the real S_{id} would run $\mathsf{Prot}_{\mathsf{id}}$, in two stages.

Stage 1 (request). Sim holds $\mathcal{A}$'s two Phase-B messages, $(\widetilde{Req}^{\mathcal{A}}, \mathsf{SR}^{\mathcal{A}}_{\mathsf{req}})$ to A_K and a mask $Y^{\mathcal{A}}$ to A_T that the in-head A_T has checked to be zero outside $\mathsf{SR}^{\mathcal{A}}_{\mathsf{req}}$, as forwarded by A_K, and sets $Req' \leftarrow \widetilde{Req}^{\mathcal{A}} \oplus Y^{\mathcal{A}}$. This is the plaintext the in-head A_T encrypted, hence, when the accepted record is the one A_T produced, the request the real S_{id} would decrypt.

Stage 2 (input). We take $\mathcal{A}$ to be the dummy adversary $\mathcal{D}$, without loss of generality [Can01], so every message of the corrupted U is an instruction of $\mathcal{Z}$. Let aux be the sequence of instructions $\mathcal{Z}$ has given $\mathcal{D}$ so far, all of which Sim has received; then $Req' = \mathcal{D}(1^\lambda, \mathsf{aux})$. Sim runs the extractor $\mathsf{E}_{\mathcal{D}}$ of Definition B.2, obtains $(\mathsf{data}', R'_{\mathsf{U}}) \leftarrow \mathsf{E}_{\mathcal{D}}(1^\lambda, \mathsf{aux}, \varepsilon, Req')$, with ε the empty tape since $\mathcal{D}$ uses no coins, and submits $(\mathsf{sid}, \mathtt{Prove}, \mathsf{id}, \bot, \mathsf{data}', R'_{\mathsf{U}})$ to $\mathcal{F}_{\mathsf{zkPoPE}}$ on behalf of U. $\mathcal{F}_{\mathsf{zkPoPE}}$ runs $\mathsf{Prot}_{\mathsf{id}}$ in head, delivers $\mathsf{out}_{\mathsf{Service}}$ to the service, which Sim releases at once, delivers the reply $Resp'$ to Sim as the corrupted caller's output, and asks Sim for the predicate. Sim leaves that request pending, encrypts $Resp'$ under $(k_{\mathsf{dec}}, IV_{\mathsf{dec}})$ as the in-head S_{id}'s response record, and continues Phases C and D against $\mathcal{A}$, its in-head attestors checking and signing whatever $\mathcal{A}$ sends, with the set $\mathsf{SR}^{\mathcal{A}}_{\mathsf{req}}$ from Phase B and the $\mathsf{SR}^{\mathcal{A}}_{\mathsf{resp}}$ that $\mathcal{A}$ supplies in Phase D.

Answering. When Phase D completes, Sim answers the pending predicate request. If $Resp' \in \mathcal{E}_{\mathsf{Prot}_{\mathsf{id}}}$ it returns the predicate that is identically 0. Otherwise it returns $\Phi^{\mathcal{A}}$, the supported predicate with position sets $(\mathsf{SR}^{\mathcal{A}}_{\mathsf{req}}, \mathsf{SR}^{\mathcal{A}}_{\mathsf{resp}})$ whose condition asserts, at every position outside them, the byte that Req' or $Resp'$ carries there. $\mathcal{F}_{\mathsf{zkPoPE}}$ then computes res, announces it and, on $res = 1$, requests a proof, which Sim answers with the π formed from the signatures its in-head attestors issued in Phase D and the values they signed. By construction $\widehat{\Phi}^{\mathcal{A}}(Req', Resp') = 1$; and when extraction succeeds, Lemma 1 gives that $\mathcal{F}_{\mathsf{zkPoPE}}$'s in-head run put Req' on the wire, so its reply is $Resp'$ and $res = \Phi^{\mathcal{A}}(\mathsf{data}', Resp') = \widehat{\Phi}^{\mathcal{A}}(Req', Resp') = 1$, and $\mathcal{F}_{\mathsf{zkPoPE}}$ records $(\mathsf{U}, \Phi^{\mathcal{A}}, \mathsf{id}, \pi)$. If $\mathcal{A}$ never completes Phase D, the predicate request stays pending and nothing is recorded, as nothing is signed in REAL. If $\mathcal{A}$ never delivers an acceptable request record, $\mathcal{F}_{\mathsf{zkPoPE}}$ is never invoked.

Two features of Definition B.2 make this construction possible inside a UC simulation: $\mathsf{E}_{\mathcal{D}}$ is straight-line, so Sim never rewinds an execution whose schedule it does not control; and its auxiliary input is quantified universally, so it accepts the instruction stream $\mathcal{Z}$ produced, whatever it is. We write E_{ext} for the event that in some session $Req' \in \mathcal{R}_{\mathsf{Prot}_{\mathsf{id}}}$ but $\mathsf{Prot}_{\mathsf{id}}.\mathsf{U}_1(\mathsf{data}'; R'_{\mathsf{U}}) \neq Req'$. For each session the auxiliary input is fixed when the extractor runs, so Definition B.2 bounds the failure probability by $\varepsilon^{\mathsf{ext}}_{\mathsf{Prot}_{\mathsf{id}},\mathcal{D}}(\lambda)$, and over q sessions

$$\Pr[E_{\mathsf{ext}}] \;\leq\; q \cdot \varepsilon^{\mathsf{ext}}_{\mathsf{Prot}_{\mathsf{id}},\mathcal{D}}(\lambda).$$

D.3 Indistinguishability: Case B (soundness)

In Case B, $\mathcal{A}$ controls U while $\mathsf{A}_K, \mathsf{A}_T, S_{\mathsf{id}}, \mathcal{V}$ are honest, $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_K]$ and $\mathcal{F}_{\mathsf{CERT}}[\mathsf{A}_T]$ run in their table-based honest-owner branch, and $\mathcal{F}_{\mathsf{US}}$ is in adversarial-network mode. We show

$$\left|\Pr[\mathcal{Z} \to 1 \mid \mathrm{REAL}_{\mathcal{RP},\mathcal{A},\mathcal{Z}}] - \Pr[\mathcal{Z} \to 1 \mid \mathrm{IDEAL}_{\mathcal{F}_{\mathsf{zkPoPE}},\mathsf{Sim},\mathcal{Z}}]\right| \;\le\; 2q \cdot \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD}}(\lambda) + q^2 \cdot 2^{-\lambda-1} + q \cdot \varepsilon^{\mathsf{ext}}_{\mathsf{Prot}_{\mathsf{id}},\mathcal{D}}(\lambda), \quad (1)$$

through games $G_0 = \mathrm{REAL}$, G_1, G_2, $G_3 = \mathrm{IDEAL}$.

Hop $G_0 \to G_1$: wire integrity. G_1 is G_0 except that S_{id} rejects any request record $(Req_{\mathsf{enc}}, T_{\mathsf{req}})$ that A_T did not produce in that session, and A_T rejects any response record $(Resp_{\mathsf{enc}}, T_{\mathsf{resp}})$ that S_{id} did not produce in that session. The two games differ only on the event E_{wire} that G_0's honest parties accept such a record in some session.

We bound $\Pr[E_{\mathsf{wire}}]$ by an adversary $\mathcal{A}'_{\mathsf{wire}}$ against $\mathsf{Exp}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD}}$. It guesses a session $s \in [q]$ and a direction, runs G_0 for $\mathcal{Z}$ and $\mathcal{A}$, simulating every honest party and functionality with keys it samples itself, except that on the guessed direction of session s it uses the game's key: the honest encryption of that direction — the A_K–A_T run of Π_{AEnc} on the request side, S_{id}'s AEAD encryption on the response side — is replaced by one $\mathcal{O}_{\mathsf{Enc}}$ query, and the honest verification — S_{id}'s on the request side, A_T's in Π_{ADec} on the response side — by $\mathcal{O}_{\mathsf{Dec}}$ queries. Since each nonce is used once, the nonce-respecting $\mathcal{O}_{\mathsf{Enc}}$ answers every redirected call. In Case B, $\mathcal{A}$ receives from the honest parties only what $\mathcal{P}_D$ receives in Π_{AEnc} and Π_{ADec} — the record (C, T), the verdict, and the keystream K_{resp} released after a successful verdict, which $\mathcal{A}'_{\mathsf{wire}}$ computes as $Resp_{\mathsf{enc}} \oplus Resp$ since it knows the plaintext it encrypted — so the simulation is faithful, and a forgery in the guessed session and direction is an $\mathcal{O}_{\mathsf{Dec}}$ query accepted on a tuple outside $\mathcal{Q}$. Hence

$$\Pr[E_{\mathsf{wire}}] \;\le\; 2q \cdot \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD},\mathcal{A}'_{\mathsf{wire}}}(\lambda).$$

Hop $G_1 \to G_2$: record-based verification. For each session s that completes Phase D write π^s for the proof its signatures determine, $(Req^s, Resp^s)$ for the request S_{id} decrypted and the response it returned, and Φ^s for the supported predicate with s's position sets asserting the disclosed bytes of $(Req^s, Resp^s)$, when $Resp^s \notin \mathcal{E}_{\mathsf{Prot}_{\mathsf{id}}}$. G_2 is G_1 except that $\mathcal{V}$ answers a query $(\mathsf{id}, P_i, \Phi, \pi^*)$ by the rule: *accept iff $P_i = \mathsf{U}$ and, for some session s with $Resp^s \notin \mathcal{E}_{\mathsf{Prot}_{\mathsf{id}}}$, $\pi^* = \pi^s$, $\mathsf{id} = \mathsf{id}^s$ and $\Phi = \Phi^s$.*

The two games differ only on the event E_{forge} that G_1's Verify and the rule disagree on some query

$$\pi^* = (\sigma^*_K, \sigma^*_T, \mathsf{id}^*, \mathsf{ssid}^*, \mathsf{SR}^*_{\mathsf{req}}, \mathsf{SR}^*_{\mathsf{resp}}, Req^*_{\mathsf{null}}, \widetilde{K}^*_{\mathsf{resp}}, \widetilde{Resp}^*_{\mathsf{enc}}).$$

Both $\mathcal{F}_{\mathsf{CERT}}$ instances accept only pairs in their sign tables, and each honest attestor signs once per session, on a message containing that session's ssid. So σ^*_K pins $(P_i, \mathsf{id}^*, \mathsf{ssid}^*, \mathsf{SR}^*_{\mathsf{req}}, \mathsf{SR}^*_{\mathsf{resp}}, Req^*_{\mathsf{null}}, \widetilde{K}^*_{\mathsf{resp}})$ to the message A_K signed in a unique session s_K, and σ^*_T pins $(P_i, \mathsf{id}^*, \mathsf{ssid}^*, \mathsf{SR}^*_{\mathsf{req}}, \mathsf{SR}^*_{\mathsf{resp}}, \widetilde{Resp}^*_{\mathsf{enc}})$ to the message A_T signed in a unique session s_T, with $\mathsf{ssid}^{s_K} = \mathsf{ssid}^* = \mathsf{ssid}^{s_T}$. Each ssid carries a nonce $r \xleftarrow{\$} \{0,1\}^\lambda$ that A_K samples fresh per session, so $s_K \neq s_T$ requires a collision among q nonces, of probability at most $q^2 \cdot 2^{-\lambda-1}$. When $s_K = s_T = s$, every component of π^* equals that of π^s, the signed messages carry $P_i = \mathsf{U}$, and $\mathcal{V}$'s parse of ssid^* enforces $\mathsf{id} = \mathsf{id}^s$. It remains to compare the two verdicts on π^s. In G_1, $\mathcal{V}$ checks that Φ's position sets are $(\mathsf{SR}^s_{\mathsf{req}}, \mathsf{SR}^s_{\mathsf{resp}})$ and evaluates Φ on $(Req^s_{\mathsf{null}}, \widetilde{Resp}^s_{\mathsf{enc}} \oplus \widetilde{K}^s_{\mathsf{resp}})$. In G_1 the record S_{id} decrypted is the one A_T produced, so $Req^s = \widetilde{Req}^s \oplus Y^s$ with Y^s zero outside $\mathsf{SR}^s_{\mathsf{req}}$ by A_T's check, and $Req^s_{\mathsf{null}} = \mathsf{zero}_{\mathsf{SR}^s_{\mathsf{req}}}\left(\widetilde{Req}^s\right) = \mathsf{zero}_{\mathsf{SR}^s_{\mathsf{req}}}(Req^s)$; likewise the record A_T accepted is S_{id}'s, so $\widetilde{Resp}^s_{\mathsf{enc}} \oplus \widetilde{K}^s_{\mathsf{resp}} = \mathsf{zero}_{\mathsf{SR}^s_{\mathsf{resp}}}(Resp^s)$. $\mathcal{V}$ therefore evaluates $\widehat{\Phi}(Req^s, Resp^s)$. Since Φ is supported, this is 1 iff Φ's sets are s's and its condition fixes, at every disclosed position, the byte $(Req^s, Resp^s)$ carries there — that is, iff $\Phi = \Phi^s$ — and rejection soundness makes it 0 whenever $Resp^s \in \mathcal{E}_{\mathsf{Prot}_{\mathsf{id}}}$. This is the rule of G_2. Hence

$$\Pr[E_{\mathsf{forge}}] \;\le\; q^2 \cdot 2^{-\lambda-1}.$$

Hop $G_2 \to G_3$. In IDEAL, $\mathcal{V}$ accepts $(\mathsf{id}, P_i, \Phi, \pi^*)$ iff $(P_i, \Phi, \mathsf{id}, \pi^*) \in \Pi_{\mathsf{sid}}$, and Sim's in-head S_{id} and A_T apply the acceptance rule of G_1. By the construction of Sim, Π_{sid} receives an entry for session s exactly when $\mathcal{A}$ completes Phase D and $Resp^s \notin \mathcal{E}_{\mathsf{Prot}_{\mathsf{id}}}$, and that entry is $(\mathsf{U}, \Phi^s, \mathsf{id}, \pi^s)$, provided the reply $\mathcal{F}_{\mathsf{zkPoPE}}$ returned is the response Sim put on the wire; conditioned on $\overline{E_{\mathsf{ext}}}$ it is, by Lemma 1. So the record of G_3 is the record of G_2. The remaining views agree as well: the in-head parties follow $\mathcal{RP}$; the response Sim encrypts is $\mathsf{Prot}_{\mathsf{id}}.\mathsf{Service}(\mathsf{sec}_{\mathsf{id}}, Req'; \cdot)$ on fresh coins, exactly the distribution of the real S_{id}'s response on the same request, independently across sessions; and the service's output $\mathsf{out}_{\mathsf{Service}}$ is released when the real S_{id} would produce it. Hence $G_2 \equiv G_3$ conditioned on $\overline{E_{\mathsf{ext}}}$, and the bound follows by summing the three events.

D.4 Indistinguishability: Cases A, C, D (zero-knowledge)

In Cases A, C and D the User is honest, and IDEAL differs from REAL in three respects: $\mathcal{F}_{\mathsf{zkPoPE}}$ answers `Verify` from its record Π_{sid}, where $\mathcal{V}$ checks the two signatures and evaluates Φ; Sim's in-head S_{id} and A_T apply the acceptance rule for wire records; and the in-head U and S_{id} run on $(Req^\star, Resp^\star)$ in place of $(Req, Resp)$. The acceptance rule has an effect only in Case A: in Cases C and D, $\mathcal{F}_{\mathsf{US}}$ is in secure-channel mode and delivers every record unaltered. The requests agree outside $\mathsf{SR}_{\mathsf{req}}$ in every session, by step 2 of Prove. When $res = 1$ the responses agree outside $\mathsf{SR}_{\mathsf{resp}}$; when $res = 0$ they may differ anywhere, but Phase D does not run, so nothing derived from the response other than its encryption appears in any view. In every game below the honest U's output $(\texttt{Executed}, Resp)$ to $\mathcal{Z}$ is the real response, as $\mathcal{F}_{\mathsf{zkPoPE}}$ delivers it in IDEAL. Everything else — the honest parties' code, the functionalities, the delivery of outputs — is identical, and the π that $\mathcal{F}_{\mathsf{zkPoPE}}$ stores is the one the in-head U assembles. We show that $\mathcal{Z}$'s distinguishing advantage is at most

$$q^2 \cdot 2^{-\lambda-1} + 2q \cdot \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD}}(\lambda) + 2q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA}\text{-}\mathcal{P}_T}_{\mathsf{D\text{-}AEAD}}(\lambda)$$

in Case A, at most $q^2 \cdot 2^{-\lambda-1} + 2q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA}\text{-}\mathcal{P}_T}_{\mathsf{D\text{-}AEAD}}(\lambda)$ in Case D, and at most $q^2 \cdot 2^{-\lambda-1} + q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}\text{-}\mathcal{P}_K}_{\mathsf{D\text{-}AEAD}}(\lambda)$ in Case C. By Theorem C.6, $\mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD}}$ and $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}\text{-}\mathcal{P}_T}_{\mathsf{D\text{-}AEAD}}$ are negligible under jPR and OTUF, and $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}\text{-}\mathcal{P}_K}_{\mathsf{D\text{-}AEAD}}$ is zero.

Verification. For each session s in which both attestors have signed, write π^s for the proof the signatures determine and Φ^s for the predicate of the session's Prove input. Let G_{rec} be REAL except that $\mathcal{V}$ answers a query $(\mathsf{id}, P_i, \Phi, \pi^*)$ by the rule: *accept iff, for some such session s run by P_i, $\pi^* = \pi^s$, $\mathsf{id} = \mathsf{id}^s$ and $\Phi = \Phi^s$.* Both attestor instances of $\mathcal{F}_{\mathsf{CERT}}$ accept only pairs in their sign tables, a semi-honest owner being uncorrupted for $\mathcal{F}_{\mathsf{CERT}}$, so the argument of hop $G_1 \to G_2$ in Section D.3 applies: unless two of the q session nonces collide, a π^* that $\mathcal{V}$ accepts equals π^s for one session s, run by P_i with $\mathsf{id} = \mathsf{id}^s$. In that session the honest U entered Phase D only after checking Φ^s on the request it sent and the response it recovered, and $\mathcal{V}$ reconstructs from π^s these two strings with the positions in $\mathsf{SR}_{\mathsf{req}}$ and $\mathsf{SR}_{\mathsf{resp}}$ set to zero. They satisfy Φ^s, and Φ is supported, so $\mathcal{V}$'s checks pass iff $\Phi = \Phi^s$. Hence

$$|\Pr[\mathcal{Z} \to 1 \mid \mathrm{REAL}] - \Pr[\mathcal{Z} \to 1 \mid G_{\mathsf{rec}}]| \;\le\; q^2 \cdot 2^{-\lambda-1}.$$

In IDEAL, Sim answers the proof request of a session once both in-head attestors have signed, so Π_{sid} holds $(P, \Phi^s, \mathsf{id}^s, \pi^s)$ for the same sessions, and $\mathcal{F}_{\mathsf{zkPoPE}}$ applies the same rule. Every game below answers `Verify` by it.

Cases A and D. In Case A, $\mathcal{F}_{\mathsf{US}}$ is in adversarial-network mode, and $\mathcal{A}$ sees the $\mathcal{F}_{\mathsf{US}}$ records $(Req_{\mathsf{enc}}, T_{\mathsf{req}})$ and $(Resp_{\mathsf{enc}}, T_{\mathsf{resp}})$, the $\mathcal{F}_{\mathsf{SC}}$ length leaks, and the proofs output to $\mathcal{Z}$. In Case D, $\mathcal{F}_{\mathsf{US}}$ is in secure-channel mode, and $\mathcal{A}$ sees the $\mathcal{F}_{\mathsf{SC}}$ and $\mathcal{F}_{\mathsf{US}}$ length leaks, the proofs output to $\mathcal{Z}$, and A_T's view: the mask $\mathsf{embed}_{\mathsf{SR}_{\mathsf{req}}, \ell_{Req}}(M_{\mathsf{req}})$ from U, $(\widetilde{Req}_{\mathsf{enc}}, \tau_{\mathsf{req}}, \mathsf{SR}_{\mathsf{req}})$ from A_K, the record $(Req_{\mathsf{enc}}, T_{\mathsf{req}})$ it computes, then $(Resp_{\mathsf{enc}}, T_{\mathsf{resp}})$, τ_{resp} and its verdict, and in Phase D $(\mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}})$, $\widetilde{Resp}_{\mathsf{enc}}$ and σ_T. Under $\mathsf{A}_T \leftrightarrow \mathcal{P}_T$ this is $\mathcal{P}_T$'s view in the session's Π_{AEnc} and Π_{ADec} runs, plus values that are functions of it and of public data. It contains both records, the only wire messages that depend on the plaintexts.

Let G_1 be G_{rec} with the acceptance rule of the simulation: S_{id} and A_T reject any record their counterpart did not produce in that session. In Case D every record is delivered unaltered, so $G_1 = G_{\mathsf{rec}}$. The secure-channel

mode of $\mathcal{F}_{\mathsf{US}}$ is needed here: $\mathcal{A}$ holds the tag material τ_{req} and τ_{resp}, and INT-CTXT bounds forgeries by $\mathcal{P}_D$ alone. In Case A, $\mathcal{A}$ may deliver altered records, and G_{rec} and G_1 differ only when one is accepted. The adversary $\mathcal{A}'_{\mathsf{wire}}$ of Section D.3, now also simulating the honest U, bounds this by $2q \cdot \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD}}(\lambda)$; its simulation is faithful because the redirected direction reaches $\mathcal{A}$ only as the record, which $\mathcal{O}_{\mathsf{Enc}}$ returns, and through the proof, which $\mathcal{A}'_{\mathsf{wire}}$ assembles from the records and the plaintexts it encrypted. In both cases we hybridize from G_1 over the $2q$ pairs (session, direction). Let $H_0 = G_1$, and let H_j run the first j pairs, in a fixed order, on the starred plaintext and the rest on the real one; $H_{2q} = \text{IDEAL}$. An adversary $\mathcal{A}'_{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}$ against IND-CCA-$\mathcal{P}_T$ (Definition C.2) picks $j \xleftarrow{\$} [2q]$, runs H_{j-1} for $\mathcal{Z}$ and $\mathcal{A}$ with locally sampled keys everywhere except on pair j, and there queries

$$\mathcal{O}_{\mathsf{Chal}\text{-}b}(n, AD, \mathsf{SR}, P_0, P_1)$$

with n the direction's nonce, P_0 the real plaintext and P_1 the starred one, $\mathsf{SR} = \mathsf{SR}_{\mathsf{req}}$ on the request side, and on the response side $\mathsf{SR} = \mathsf{SR}_{\mathsf{resp}}$ when $res = 1$ and $\mathsf{SR} = [\ell_{Resp}]$ when $res = 0$; in every case P_0, P_1 have length ℓ and agree outside SR. The oracle returns $\mathcal{P}_T$'s view $(M, \widetilde{C}, \boldsymbol{\tau}, C, T)$ of Π_{AEnc} on P_b. On the request side this is $(\mathsf{embed}_{\mathsf{SR}_{\mathsf{req}},\ell_{Req}}(M_{\mathsf{req}}), \widetilde{Req}_{\mathsf{enc}}, \tau_{\mathsf{req}}, Req_{\mathsf{enc}}, T_{\mathsf{req}})$; on the response side (C, T) is S_{id}'s record and $\boldsymbol{\tau} = \tau_{\mathsf{resp}}$ is what A_T receives in Π_{ADec}, the other components being discarded. $\mathcal{A}'_{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}$ forwards to $\mathcal{A}$ the part of this view the case prescribes, and completes the session from it: the Phase-D values $Req_{\mathsf{null}} = \mathsf{zero}_{\mathsf{SR}_{\mathsf{req}}}(P_b)$ and $\widetilde{Resp}_{\mathsf{enc}} \oplus \widetilde{K}_{\mathsf{resp}} = \mathsf{zero}_{\mathsf{SR}_{\mathsf{resp}}}(P_b)$ are the same for $b = 0, 1$, and $\widetilde{K}_{\mathsf{resp}} = \mathsf{zero}_{\mathsf{SR}_{\mathsf{resp}}}(C \oplus P_b)$ is computable from the returned C and either plaintext. When $b = 0$ the view is that of H_{j-1}, when $b = 1$ that of H_j, and $\mathcal{A}'_{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}$ outputs $\mathcal{Z}$'s bit. Summing over j,

$$|\Pr[\mathcal{Z} \to 1 \mid G_1] - \Pr[\mathcal{Z} \to 1 \mid \text{IDEAL}]| \;\le\; 2q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}_{\mathsf{D\text{-}AEAD},\mathcal{A}'_{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}}(\lambda).$$

Case C. $\mathcal{F}_{\mathsf{US}}$ is in secure-channel mode, so $\mathcal{A}$ sees no wire bytes and delivers every record unaltered; the acceptance rule of the simulation changes nothing. Its view consists of the $\mathcal{F}_{\mathsf{SC}}$ and $\mathcal{F}_{\mathsf{US}}$ length leaks, the proofs output to $\mathcal{Z}$, and A_K's view: the handshake state $(r, \kappa, k_{\mathsf{enc}}, k_{\mathsf{dec}}, IV_{\mathsf{enc}}, IV_{\mathsf{dec}})$; $\widetilde{Req}$, $\mathsf{SR}_{\mathsf{req}}$ and the material $K_{\mathsf{req}}, \tau_{\mathsf{req}}, \widetilde{Req}_{\mathsf{enc}}$ it computes in Phase B; $|Resp_{\mathsf{enc}}|$, τ_{resp}, the verdict and K_{resp} in Phase C; and $(\mathsf{SR}_{\mathsf{req}}, \mathsf{SR}_{\mathsf{resp}})$, Req_{null}, $\widetilde{K}_{\mathsf{resp}}$, σ_K in Phase D. Every component other than $\widetilde{Req}$ is a function of κ, of public data, and of $\widetilde{Req}$; the proof π is a function of $\widetilde{Req}$, κ, public data and $\widetilde{Resp}_{\mathsf{enc}} = \mathsf{zero}_{\mathsf{SR}_{\mathsf{resp}}}(Resp_b \oplus K_{\mathsf{resp}})$, which for $res = 1$ is the same for $Resp_0 = Resp$ and $Resp_1 = Resp^\star$. So G_{rec} and IDEAL differ only in $\widetilde{Req} = Req_b \oplus \mathsf{embed}_{\mathsf{SR}_{\mathsf{req}},\ell_{Req}}(M_{\mathsf{req}})$ with $Req_0 = Req$ and $Req_1 = Req^\star$, per session. This is the IND-CPA-$\mathcal{P}_K$ experiment (Definition C.1) with $\mathsf{SR} = \mathsf{SR}_{\mathsf{req}}$, and a hybrid over the q sessions gives the bound $q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA\text{-}}\mathcal{P}_K}_{\mathsf{D\text{-}AEAD}}(\lambda)$, which is 0 by Theorem C.6: $\widetilde{Req}$ agrees with Req_b outside $\mathsf{SR}_{\mathsf{req}}$, where Req_0 and Req_1 coincide, and is uniform on $\mathsf{SR}_{\mathsf{req}}$ under the one-time pad M_{req}, whichever b.

D.5 Wrap-up

For each admissible corruption pattern the distinguishing advantage of $\mathcal{Z}$ is bounded by the corresponding expression above: $2q \cdot \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD}} + q^2 \cdot 2^{-\lambda-1} + q \cdot \varepsilon^{\mathsf{ext}}_{\mathsf{Prot}_{\mathsf{id}},\mathcal{D}}$ in Case B, $q^2 \cdot 2^{-\lambda-1} + 2q \cdot \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD}} + 2q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}_{\mathsf{D\text{-}AEAD}}$ in Case A, $q^2 \cdot 2^{-\lambda-1} + 2q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}_{\mathsf{D\text{-}AEAD}}$ in Case D, and $q^2 \cdot 2^{-\lambda-1}$ in Case C. By Theorem C.6, $\mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathsf{D\text{-}AEAD}}$ and $\mathsf{Adv}^{\mathsf{IND\text{-}CCA\text{-}}\mathcal{P}_T}_{\mathsf{D\text{-}AEAD}}$ are negligible under jPR and OTUF; $\varepsilon^{\mathsf{ext}}_{\mathsf{Prot}_{\mathsf{id}},\mathcal{D}}$ is negligible because $\mathsf{Prot}_{\mathsf{id}}$ is request-extractable. Extraction is invoked only against a corrupted User. This establishes Theorem 5.1. □