NOTRY: Deniable messaging with retroactive avowal

Faxing Wang
University of Melbourne
wang39@student.unimelb.edu.au

Shaanan Cohney
University of Melbourne
cohneys@unimelb.edu.au

Riad Wahby
Carnegie Mellon University
riad@cmu.edu

Joseph Bonneau
a16z Crypto Research & New York University
jbonneau@gmail.com

ABSTRACT

Modern secure messaging protocols typically aim to provide deniability. Achieving this requires that convincing cryptographic transcripts can be forged without the involvement of genuine users. In this work, we observe that parties may wish to revoke deniability and *avow* a conversation after it has taken place. We propose a new protocol called Not-on-the-Record-Yet (NOTRY) which enables users to prove a prior conversation transcript is genuine. As a key building block we propose *avowable designated verifier* proofs which may be of independent interest. Our implementation incurs roughly 8× communication and computation overhead over the standard Signal protocol during regular operation. We find it is nonetheless deployable in a realistic setting as key exchanges (the source of the overhead) still complete in just over 1ms on a modern computer. The avowal protocol induces only constant computation and communication performance for the communicating parties and scales linearly in the number of messages avowed for the verifier—in the tens of milliseconds per avowal.

KEYWORDS

proof of non-knowledge, deniable messaging, AKE, avowal

1 INTRODUCTION

Imagine Mallory wrongfully accuses Alice and Bob of a heinous crime. Alice and Bob have an alibi attested in their chat history, but alas! They communicated using the Off-the-Record protocol [12] (OTR), which is cryptographically deniable. They are therefore unable to satisfactorily prove their innocence to the public!

In this paper, we propose a new protocol called Not-On-The-Record-Yet (NOTRY), which preserves the benefits of deniable protocols while enabling Alice and Bob to jointly *avow* the contents of a conversation should it later become necessary. NOTRY aims to provide the best of both worlds between deniable protocols (that prevent Alice or Bob from individually proving what was said to a third party) and *non-repudiable* protocols where each message is accompanied by such a proof.

Deniable protocols are commonplace. The widely used Signal [1, 55] and recent Messaging Layer Security (MLS) [2, 4] protocols both aim to provide deniability—which comes in various flavors. Such variants include transcript deniability (neither party can prove what was said) and participation deniability (neither party can prove who participated at all). Protocols achieve deniability by providing a construction that allows any party to forge a cryptographically valid conversation transcript, without knowledge of the private keys that the participants used to mutually communicate. Herein we present a new variant of deniable communications—with a twist: at some future time t_1 after their initial exchange, Alice and Bob may retroactively *avow* their transcript. That is, Alice and Bob may disseminate additional values showing that a unique version of the transcript is correct (and hence that any other claims are forgeries including those generated prior to t_1).

Beyond the exonerating evidence example provided above, there are many possible scenarios in which two communicating parties might later wish to avow a conversation that they initially intended to remain off the record. A transcript may contain evidence useful to a proceeding in which the interests of the two parties are aligned (e.g.; co-inventors who wish to prove first-to-invent in patent proceedings, an alibi for a crime, or a journalist and source proving the authenticity of a conversation after events have elapsed).

1.1 Proofs of non-knowledge

The essential technical idea underlying our work is a trick to prove *non-knowledge* of a discrete log relationship. Given a value X and two bases g, h chosen such that the discrete log relationship between them is unknown, proving knowledge of the discrete logarithm of X to the base g serves as a proof of non-knowledge of the discrete logarithm of X to the base h. Symmetrically, proving knowledge of the discrete logarithm of X to the base h proves non-knowledge of X to the base g. This works because if the discrete logarithm of X were known to both bases, this would enable computing of the discrete log of h to the base g, therefore the discrete log of X may be known to at most one of the bases g, h. The two bases can be chosen using a *nothing up my sleeve* technique such as hashing a constant value in two different ways to produce two pseudorandom group elements.

A similar idea has been used in protocols for *oblivious transfer* (OT)[5, 36, 56, 61]. For example, the sender in a classic 1-out-of-2 OT chooses two bases g, h such that the receiver doesn't know the discrete log relation between them. The receiver then generates two public keys K and $K' = \frac{K}{h}$. The sender is convinced that the receiver can't know the private key (discrete log) of both keys (to the base g) without learning a discrete log relation between g and h, and therefore can only decrypt one of two messages.

This works effectively as a proof (to the sender) of non-knowledge of one of the two private keys. In this work, we generalize this

Proceedings on Privacy Enhancing Technologies 2024(2), 391–411
© 2024 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2024-0056

idea and provide an explicit description (where previously it was implicit). We extend the idea to proofs-of-non-knowledge in a publicly verifiable setting (whereas in OT the proofs are effectively designated-verifier). We also apply this idea in a new context, avowal of (formerly) deniable proofs. Consider classic designated verifier proofs that prove either knowledge of some witness w for a statement S or knowledge of a secret key sk_V. We can transform this proof by proving the following:

(knowledge of w AND the discrete log of X to the base g) OR (knowledge of sk_V AND the discrete log of X to the base h).

If nothing is known about X, then this serves as a designated verifier proof, as it may have been satisfied either by knowing the genuine witness w or the verifier's key sk_V. However, if a separate proof establishes that the discrete log of X is known to either base, this establishes which half of the disjunction was satisfied and reveals if the original designated verifier proof is real or a forgery.

We use an extension of this technique during NOTRY's authenticated key exchange, in which Alice and Bob mutually create a value $X = AB$ such that its discrete log to the base g is secret-shared between the two parties (e.g. $AB = g^{\gamma+\delta}$ for values γ, δ chosen by Alice and Bob respectively). If desired, the two parties can later collaborate to prove knowledge of this discrete logarithm and revoke deniability of the key exchange.

1.2 Contributions

With NOTRY we present the first Deniable Authenticated Key Exchange (DAKE) satisfying the security properties described by Unger [66] while additionally supporting *retroactive avowal*. Our technical contributions are as follows:

- We first develop a new primitive: *avowable designated verifier proofs*. As with standard designated verifier proofs, they can be forged by a party with a specified public key (the designated verifier) and hence are not convincing to other parties. However, with avowable designated verifier proofs the prover later convince a verifier that a previously generated proof is genuine and not a forgery. We formally define the notion of an avowable designated verifier proof as well as providing a concrete instantiation.
- We construct a modified DAKE that incorporates our avowable designated verifier proof construction, leading to a secure messaging protocol with retroactive avowal. Our proposal (NOTRY) is the first DAKEA (DAKE with avowal) protocol. We assume that both parties store avowal secrets and verifiers store a genuine log of ciphertext transcripts to be avowed. Without authentic ciphertext it is trivial for parties to retroactively create a fake conversation and avow it. Designing an efficient and secure logging protocol is out of our scope.
- We prove the security of our construction using the Universal-Composability framework [16] under a special random oracle model. The proof is conducted under the standard decisional Diffie-Hellman hardness assumption (noting that our protocol is therefore *not* quantum secure). We show that NOTRY guarantees a strong form of deniability, online deniability [33, 68], and also demonstrate that NOTRY offers standard AKE security properties of AKE.
- We implement NOTRY and evaluate its performance. In comparison to Signal [55, 59], clients incur an 8× times communication overhead and an 8× times computational overhead per key exchange—a manageable performance hit in the face of likely future improvements. When parties wish to avow a message, both they and the *judge* pay further computation and communication costs to run the avowal protocol.
- We further show that our avowal protocol scales effectively to an entire transcript. Parties are free to avow a stream of messages in a conversation without incurring communication overhead above that of avowing a single message. They only need to operate an extra scalar addition for each additional message avowal. The computation and communication performance overhead of the designated verifier grows linearly with the number of messages to be avowed for the verifier, with the slope of 0.006 for computation and receiving extra 32 bytes each time for avowing one more message.

The remainder of the paper proceeds as follows: We commence in Section 2 with a review of work relevant to our design space, which includes message franking schemes and work on retroactivity in cryptography. In Section 3 we introduce all the primitives needed to build our scheme. This proceeds with reintroducing designated verifier proofs (DVs) for the unfamiliar reader, before proceeding on our novel construction of avowable designated verifier proofs (ADVs)—allowing a prover to render a proof verifiable at some point after they first generate it. In Section 3.4 we extend ADVs by introducing mutually avowable DVs, whereby it takes two cooperating provers to avow a message transcript to a third-party verifier. We are then in Section 3.4 able to introduce our overall deniable messaging scheme with retroactive avowal (NOTRY), deferring its security proofs until Section 5. We conclude with implementing and evaluating the protocol in Section 6 followed by a few closing remarks.

2 RELATED WORK

Our work draws from several similar notions to that of cryptographic avowal—*disavowal* (its mirror image) and message franking (a one-sided complement to avowal). Here we highlight works that relate across two axes: similar cryptographic notions, and deniable messaging protocols on which we build.

2.1 Avowal, Disavowal, and Franking

Chaum[24] defined cryptographic *disavowal* in his work on zero-knowledge signatures —wherein a signer could convince others under zero-knowledge that a signature does not correspond to the signer's public key and purported message. Cryptographic avowal is also closely related to *message franking* [41, 65]. Message franking schemes allow a single party in a conversation to prove the authenticity of part of a conversation, typically only to one specific third party (the judge). Such schemes are useful for abuse management in encrypted communication systems, as they can provide a method for a message recipient to report content to a platform hosting the otherwise encrypted messaging service. Our scheme differs in two ways: 1) both parties to a conversation must agree to avow a conversation and 2) they can choose any third-party as a judge. This vitiates its usefulness as an abuse reporting mechanism but leads to the other use cases we described in Section 1.

2.2 Retroactivity in Signature Schemes

There are several other primitives of a similar flavor—providing the ability for parties to change the epistemic status of signatures *ex post facto*, which we here review.

Park and Sealfon introduced the notion of *claimable ring signatures* [57]. Traditional ring signatures provide a form of deniability by allowing a signer to choose an arbitrary set of public keys and prove that a message was signed by some key from that set (without revealing which key). Claimable ring signatures allow the signer to later claim the signature by proving that their specific key was the one used to sign, similar to our notion of revoking deniability after the fact. The corollary notion of unclaimable ring signatures also exists in which it is not possible to provably claim who signed.

Much like ring signature schemes, some threshold signature sch-emes offer accountability in that the signature reveals which key shares were used to sign. Others may offer privacy in that signatures are indistinguishable regardless of which shares were used. Boneh and Komlo [11] proposed threshold signature schemes with private accountability by introducing a separate accountability key that can be used to compute (and prove) which shares were used to sign, similar to our notion of avowal.

Chaum and van Antwerpen introduced *undeniable signatures* [25]. Undeniable signatures cannot be verified without online interaction with the original signer (though the verification protocol does not provide transferable proof). Effectively, the signer can "disavow" a past signature by refusing to verify it, meaning signatures are effectively deniable until avowed. This is somewhat similar to the goal of proofs with retroactive avowal, although (among other differences) we are working in a two-party setting and support public avowal.

2.3 Secure Messaging and Deniability

Modern secure messaging protocols, such as Signal, possess a variety of cryptographic properties, many derived from the extensions on a key-exchange protocol (Diffie-Hellman [32]). The most basic required additional property is authentication, achieved in later authenticated key exchange (AKE) protocols [7, 31, 46, 51]. Modern protocols also possess more sophisticated properties such as composability [6, 8, 14, 20, 29, 52], security under distinct adversarial powers [20, 27, 50], forward secrecy [27, 46], key confirmation [29, 38], and repudiability/deniability [34].

Deniability allows users to convince a judge that they never had such a conversation. It works by providing an easy mechanism for a third-party to generate a forgery that a judge would be unable to distinguish from a legitimate transcript.

Deniability is of particular interest because it directly contrasts with the otherwise desired property of *non-repudiabiliy*—like past work on franking, our contribution suggests that in the context of secure messaging, both properties may be desirable depending on local context.

Early work explored potential deniable protocols [12, 13, 47, 48] but lacked a corresponding formalism. Di Raimondo et al. [30] formally introduced the notion of deniable authenticated key exchange protocol (DAKE). Deniability now spans an independent line of research [34, 35, 44, 71] and is a standard feature for secure messaging applications, such as Signal and OTR. Considerable work has gone into improving the performance characteristics of DAKEs [30, 33, 43, 55, 63, 66, 70]. Finally we point to two works that we particularly draw on for our extensions: Walfish [69] who devised a DAKE protocol which provides deniability, forward secrecy, adaptive secure, and without a trusted third party, and Unger [66] who presented protocols enhanced against insider threats and compatible with group messaging.

3 BUILDING BLOCKS

3.1 Notations

Let q be a prime and $\mathbb{G}$ be a cyclic group of order q. We use g and h to denote two independently sampled generators of $\mathbb{G}$ (that is, the discrete log relationship of g and h is unknown). Suppose Alice holds a pair of identity keys, the corresponding public key pk and private key sk are denoted as $k_A = (pk_A, sk_A)$. For i-th round key exchange protocol, we denote γ_i *and* δ_i to be secrets independently chosen by each party and A_i *and* B_i to be the corresponding public exponents to the base h. We denote the secret for avowal as α and β respectively.

We write a uniform random number r sampled from set U as $r \xleftarrow{\$} U$. We use $X \leftarrow Y$ to denote X being set to Y and $\approx_c$ to denote computational indistinguishability. Our security parameter overall is denoted as λ. We use $||$ to denote message concatenation, as in $m_1||m_2$. Finally, σ denotes a signature and K denotes an established symmetric key.

3.2 Designated Verifier Proofs

To develop our construction we start with Jakobsson, Sako, and Impagliazzo's idea of designated verifier (DV) proofs. They used a disjunctive statement to provide deniability [42].[1] To illustrate this notion we introduce two parties, Alice (the prover) who possesses a statement S, and a Judge, Judy (the verifier).

Given a statement S to be proven to a verifier in zero knowledge, Jackobsson et al. proposed issuing a proof of the modified statement:

$$S' = \{Either\ S\ is\ true\ or\ I\ know\ Judy's\ private\ key\} \tag{3.1}$$

We will abbreviate this construction here:

$$S' = S \vee \mathsf{Know}(sk_J) \tag{3.2}$$

A proof of the compound statement S' is only convincing to Judy: If Judy is confident that nobody else knows her private key sk_J (and that she did not compute the proof), then she knows the second clause is false and therefore S is true.

For anyone other than Judy, it is unclear which side of the disjunction is true—it is possible that *either* S is true or that Judy herself created a valid proof of S' by satisfying the second clause.

[1] Another approach to constructing signatures with the designated-verifier property is *chameleon signatures* [49], which uses a standard hash-and-sign construction but with a chameleon hash function whose trapdoor is known by the intended verifier, providing deniability if the verifier attempts to transfer the signature to another party.

Therefore, the prover has deniability—the prover can always claim to an outsider that Judy generated the proof whether or not S is true, and the outsider will be unable to tell.

3.3 Avowable designated verifier proofs

We build on DV proofs to introduce what we believe is a novel tool: *avowable designated verifier* proofs (ADV proofs). We later use this to develop NOTRY, however ADV proofs may be of independent interest for other applications.

Given a statement S, an ADV proof $\bar{\pi}$ is a proof of a modified statement $\bar{S}$ with two key properties:

(1) at time t_0 no party, not even the judge, can ascertain the truth of S from $\bar{S}$ alone and
(2) at some future time t_1, the generator of the proof can complete a protocol such that the judge can validate that $\bar{S}$ attests to the truth of S.

Note that it is not enough to merely provide an unconditional (non-designated-verifier) or direct proof that S is true. Providing an unconditional proof would not preclude that previous proofs of S were created as forgeries.

A one-time ADV scheme: A strawman approach would be to prove that *nobody* knows Judy's private key. Assume that Judy's public key is a group element of the form $pk_J = g^x$ and Judy's private key is $sk_J = x$. If Judy chooses her public key pseudorandomly as $pk_J = H(y)$ for some pre-image y, then under the discrete-log assumption revealing y proves that nobody knows the corresponding private key sk_J and therefore any proofs issued to pk_J were genuine. However, this scheme means that Judy must use a new key for every proof, and the key must be chosen by the prover in the case of genuine proofs.

A candidate scheme: Instead, we allow the prover to create a new proof-specific statement P which can later be easily proven to be true or false. Given such an auxiliary statement P, our ADV proof scheme works by creating a proof of the following statement:

$$\bar{S} = (S \wedge P) \vee (\mathsf{Know}(sk_J) \wedge \neg P) \tag{3.3}$$

Revealing the veracity (or falsehood) of P then makes it clear which half of the disjunction was satisfied and hence whether such a proof was created by an honest prover or by Judy. Note that in either case, the proof creator must choose P and retain a witness to either prove P (in the case of the honest prover) or prove $\neg P$ (in case of the forger). The construction symmetrically enables both avowing an honestly created proof and avowing a forgery.

Based on the candidate scheme, we provide a formal definition of the security properties of an ADV in Section 3.4 and proofs in Section 5.4.1 (deferring them for overall clarity of exposition).

3.3.1 ADV Functions. An ADV proof scheme consists of the following five functions:

Setup(λ) generates public parameters which can be used concurrently and repeatedly for multiple ADV proofs. Separately, we assume a PKI to establish public/private key pairs for all parties and a designated judge.

Gen($\bar{S}, \bar{W}$) outputs an ADV proof $\bar{\pi}$ of the disjunction $\bar{S}$ with the witness $\bar{W} \in \{(w, w_P), (sk_J, w_{\bar{P}})\}$, where w for the statement S, sk_J for the knowledge of *Judge*'s private key, and w_P for the statement P whereas $w_{\bar{P}}$ for $\neg P$.

Avow(P, W_P) generates the avowal proof $\bar{\pi}'$ by (dis)proving an auxiliary statement P for telling which clause in $\bar{S}$ was proved. The witness W_P can *either* be w_p to avow $\bar{S}$ was satisfied via S *or* $w_{\bar{P}}$ to avow S was satisfied via the knowledge of sk_J.

Verify($\bar{\pi}, \bar{S}$) asserts that an ADV proof $\bar{\pi}$ is a valid proof for an accompanying ADV statement $\bar{S}$.

Judge($\bar{\pi}', P$) verifies the avowal proof $\bar{\pi}'$ and asserts the validity of the avowal claim.

3.3.2 Constructing ADV Proofs.

Discrete-log ADV construction: For Σ-protocols, a compelling choice for P is knowledge of the discrete log x of a value $y = g^x$ (in an appropriate group). There is a classic Schnorr Σ-protocol for proving knowledge of this discrete-log relationship [64] that one can combine with Σ-OR proofs [26] to create an efficient proof of S'. The Schnorr protocol does not permit us to *directly* prove $\neg P$ in a cyclic group: given the construction of y (raising the base g to the power of x) it is clear that group element y has a unique discrete log to the base g. Instead, we utilize another generator h sampled independently, ensuring that nobody knows the discrete log relationship between g and h.[2] Proving knowledge of the discrete log of y to the base h suffices to show that the discrete log of y to the base g is unknown. If the discrete log of y was known to both bases g and h, this would enable easily computing the discrete log of g to the base. This yields an efficient candidate for P when constructing Σ-protocols: proof of knowledge of an element y to either the base g (equivalent to proving P) or to the base h (equivalent to proving $\neg P$).

Alternately, if g and h are known to generate subgroups of a group G which are disjoint modulo the identity element, and the *subgroup membership problem* is assumed to be hard, then proving knowledge of the discrete log of y to the base g is an unconditionally sound proof that the discrete log of y to the base h is unknown (since it will not exist). However, the discrete log assumption is still required for privacy.

Hash-based ADV construction: An alternative, simple construction that might offer efficiency benefits in some circuit-based proof systems utilized preimages of a collision-resistant hash function. Specifically, P might state that given a value b, there exists a preimage $b = H(a)$ under hash function H such that a has odd parity. The corresponding $\neg P$ simply states that a has even parity. This statement can be proven true or false by revealing a (or if necessary this could be proved in zero-knowledge). Note that if two values a and a' are known such that $b = H(a) = H(a')$ (possibly with differing parity), this would be a hash collision. Thus, only one of $P, \neg P$ will be provable assuming the hash function remains collision-resistant.

3.4 Mutual ADVs

Finally, the full NOTRY protocol requires an extension to ADV proofs which enables *mutual avowal*, in which both Alice and Bob's

[2]Given a hash function H which outputs pseudorandom generators of a group $\mathbb{G}$, g and h can be chosen as $g = H(0)$, $h = H(1)$, for example.

cooperation is needed to avow the proof. We term such a proof a *mutual-ADV proof* or MADV.

The security goal of MADV avowal proof is to convince the *Judge* that Alice and Bob together do have the witness w to an ADV statement $\bar{S}$ while revealing no more information to the *Judge*. This means that MADV proof avowal is a zero-knowledge proof of knowledge (ZKPoK). Since this paper centers on avowal proofs, security definitions and corresponding proofs of ADV proofs are deferred to appendix A.

Definition: Formally, a secure MADV avowal proof $\bar{\pi}'$ against the MADV proof $\bar{\pi}$ and statement $\bar{S}$ consists of a triple of algorithms (**Setup**, **Avow**, **Judge**) with the following properties:

(1) *Completeness:* Any prover who generated an MADV proof $\bar{\pi}$ can construct the corresponding MADV proof avowal $\bar{\pi}'$ such that:

$$\Pr\left[1 \leftarrow \mathit{Judge}(\bar{\pi}, \bar{S}) \;\middle|\; \bar{\pi}' \leftarrow \mathit{Avow}(P, W_P)\right] = 1 \tag{3.4}$$

Completeness indicates that *Judge* always accepts the MADV avowal proof $\bar{\pi}'$ from an honest prover.

(2) *Knowledge Soundness*: There exists a *PPT* extractor $\mathcal{E}$ which, given two transcripts of avowal proof generation $\mathcal{T}_1$ and $\mathcal{T}_2$, can extract a valid witness:

$$\Pr\left[1 \leftarrow \mathit{Judge}(\bar{\pi}', P) \;\middle|\; \begin{array}{l} W_P' \leftarrow \mathcal{E}(1^\lambda, \mathcal{T}_1, \mathcal{T}_2) \\ \bar{\pi}' \leftarrow \mathit{Avow}(P, W_P') \end{array}\right] = 1 \tag{3.5}$$

where W_P' is the extracted witness. Special soundness implies that any party who generates a valid ADV proof $\bar{\pi}$ must gain the knowledge of W_P to P.

(3) *Avowal Soundness*: For any *PPT* adversary $\mathbb{A}$, given a MADV proof $\bar{\pi}$, it is unable to produce a valid avowal proof:

$$\Pr\left[1 \leftarrow \mathit{Judge}(\tilde{\bar{\pi}}', P) \;\middle|\; \tilde{\bar{\pi}}' \leftarrow \mathbb{A}(1^\lambda, \bar{\pi})\right] \leq \mathsf{negl}(\lambda) \tag{3.6}$$

where *negl* is a negligible function.

(4) *Honest Verifier Zero-Knowledge*: There exists a *PPT* simulator $\mathbb{S}$ that generates an MADV avowal proof $\bar{\pi}_{\mathbb{S}}'$ without knowledge of the witness w such that $1 \leftarrow \mathit{Judge}(\bar{\pi}_{\mathbb{S}}')$. $\bar{\pi}_{\mathbb{S}}'$ holds that:

$$\bar{\pi}_{\mathbb{S}}' \approx_c \bar{\pi}' \tag{3.7}$$

After defining a secure MADV avowal, we proceed to provide an MADV construction. Based on our discrete-log ADV construction, our construction of MADV begins with Alice and Bob generating a value $AB = h^{\gamma+\delta}$. The value $\gamma + \delta$ is a shared secret between Alice and Bob; let γ and δ be individual secrets belonging to Alice and Bob respectively. Generating the shared secret is easy: Alice generates γ randomly and sends $A = h^\gamma$ to Bob, who generates δ randomly and sends $B = h^\delta$ back to Alice. Both sides can compute $AB = A \cdot B = h^\gamma \cdot h^\delta = h^{\gamma+\delta}$

Recall that proving non-knowledge of a discrete log to base g is achieved by demonstrating knowledge of the discrete log of the same value to another base h where g and h are two independently sampled generators of $\mathbb{G}$. Let h be the generator used to produce the shared secret. Thus, Alice and Bob can use the mutually created AB as the value in our ADV construction, making it an MADV as avowal will require both Alice and Bob's participation to prove knowledge of the discrete log of AB to the base.

Note that if either party is malicious, they can permanently prevent future avowal. For example, Alice can choose $A = g^\gamma$ for a random γ, which will prevent ever avowing that the discrete log of AB to the base h is known. This simple attack could be prevented if both sides include a zero-knowledge proof of A, B respectively to the base h (possibly a designated verifier proof for the other party), but this doesn't prevent either party from deleting their avowal share (α or β) immediately after the handshake completes.

We foreshadow that in NOTRY, Alice and Bob each generate an MADV proof attesting to knowledge of their own secret key. Specifically, Alice and Bob provide proof of the below statements, respectively:

$$\bar{S}_A \leftarrow [\mathrm{dlog}_h A \vee \mathrm{dlog}_h B] \wedge [\mathrm{dlog}_g pk_A \vee \mathrm{dlog}_g AB] \tag{3.8}$$

$$\bar{S}_B \leftarrow [\mathrm{dlog}_h A \vee \mathrm{dlog}_h B] \wedge [\mathrm{dlog}_g pk_B \vee \mathrm{dlog}_g AB] \tag{3.9}$$

To see why this is convincing, take Alice's statement (Equation 3.8). For an honest Bob, who has sampled $B = h^\delta$, he can be sure Alice could not prove $\mathrm{dlog}_h B$, and hence Alice must know $\mathrm{dlog}_h A$. This means that $\mathrm{dlog}_g AB$ is unknown, and hence Alice must know both $\mathrm{dlog}_g pk_A$ (authenticating her) and $\mathrm{dlog}_h A$ (meaning her contribution to the avowal secret is well-formed). A symmetric argument applies to Alice's reasoning about Bob's proof.

Observe that this construction is still deniable—a forger can first randomly generate $A = h^\gamma$, then $AB = g^\chi$ for a random χ and derive $B = AB/A$. This forger can then satisfy both Equation 3.8 and Equation 3.9 by proving $\mathrm{dlog}_h A$ on the left side (using knowledge of γ and $\mathrm{dlog}_g AB$ on the right side (using knowledge of χ). Note that this forger does not need to know either secret key.

Conversely, avowal works by the parties jointly proving knowledge of $\mathrm{dlog}_h AB$, using the shared secret value $\gamma + \delta$. This rules out that the simulator was used, as the simulator (having chosen AB with a known discrete log to the base g), cannot prove knowledge of $\mathrm{dlog}_h AB$.

There are two possible flavors of avowal: public and designated verifier avowal. Depending on different applications, parties can choose to avow to a designated *Judege* for partial deniability (avowed transcripts are still deniable for anyone other than the *Judge*) or publicly avow and completely give up deniability. Constructions are similar for both types of avowal, using either an unconditional proof-of-knowledge of $\mathrm{dlog}_h AB$ for public avowal or its corresponding designated verifier version for designated-verifier avowal.

3.5 Signatures of Knowledge

Mutual ADVs alone are not sufficient to build an avowable key exchange protocol, as they are vulnerable to meddler-in-the-middle (MITM) attacks. A standard solution to address MITM attacks in key exchange protocol is to ask parties to include signatures on their messages. However, classic digital signatures will undermine deniability as they provide non-repudiable evidence that a specific private key was used to sign.

To build a deniable authenticated key exchange protocol with retroactive avowal (DAKEA) defending against MITM attack, we instead use *signatures of knowledge* [23]. Unlike classical digital signature schemes which require a specific secret key, computing a signature of knowledge (SoK) requires knowing a witness w for some statement $x \in L$ for an NP language L. Specifically, NOTRY requires parties to compute a signature of knowledge using a witness for the MADV proof statement, like eq. (3.8), on a message which commits to all the public information and the prefix of the transcript.

Definition 1. A *signature of knowledge* scheme for message space $\mathbb{M}$ and an NP language L decided by a Turning Machine M_L is defined by three polynomial-time algorithms (**Setup**, **Sign**, **Verify**)

Setup$(\lambda) \to$ pp: produces a set of public parameters pp.

Sign$(m, w, x, sk) \to \sigma$: generates a SoK σ on message m by (pp, L, x, w) with sk if M_L decides w for statement $x \in L$ such that $M_L(x, w) = 1$, otherwise outputting $\perp$.

Verify$(m, x, \sigma, pk) \to$ Accept/Reject: verifies the signature σ on message m for statement x with pk.

A signature of knowledge has three following security properties:

Correctness: If a signature σ is produced using a valid witness, then it should be accepted by a verifier with overwhelming probability.
Simulatability: By checking a signature, a verifier learns nothing beyond that the message is signed properly and the statement is true. This means that the signature of knowledge reveals nothing about the witness which is utilized to generate the signature.
Extraction: guarantees that a party who can create a valid signature must “know” a witness w for the statement x, in the sense that an extractor exists which can output a witness given access to the signer. In other words, a signature of knowledge is also a proof of knowledge issued by a signer which indicates that she knows the witness to the target statement.

We use this primitive by setting the message m to be signed as the corresponding key exchange protocol messages and letting the statement P be the ADV disjunction $\bar{S}$. Therefore, by using the discrete-log ADV construction in Section 3.3.2, a concrete SoK can be constructed. Each ADV prover generates a *Schnorr Signature* [64] with its secret key and sets the message m to be an instantiation of the ADV statement $\bar{S}$.

4 DESIGN OF NOTRY

Our NOTRY protocol consists of two sub-protocols: NOTRY-Kex and NOTRY-Avow. Like other (D)AKEs, we design our NOTRY-Kex around the Diffie-Hellman protocol with signatures for authentication. We also include a ratcheting scheme in NOTRY-Kex-Ratchet to enable future secrecy in the context of ongoing messaging—typical among secure messaging protocols [12].

4.1 NOTRY-Kex

Based on the above constructions and primitives we propose our NOTRY DAKEA key exchange protocol NOTRY-Kex. Before starting NOTRY-Kex, we assume that the authority properly sets up public parameters and identity tokens. Initially, Alice and Bob register their key pairs with a server (similar to Signal’s coordinating server or a CA). NOTRY-Kex depicted in Figure 1 proceeds as follows:

First, Alice starts the protocol by randomly selecting an ephemeral secret $\gamma \in \mathbb{Z}_q$. She sends Bob the public exponent $A \leftarrow h^{\gamma}$ as with DH protocol.

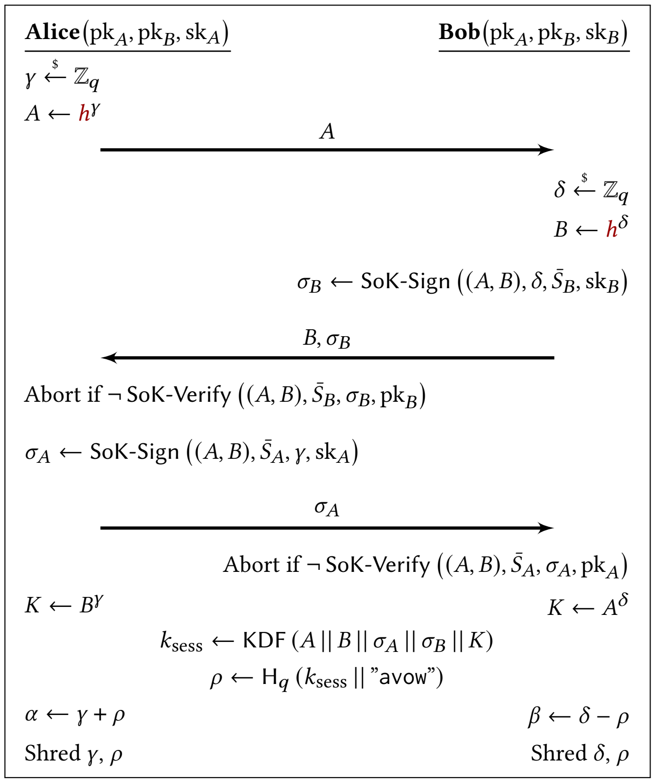

$\text{Alice}(\text{pk}_A, \text{pk}_B, \text{sk}_A)$
$\text{Bob}(\text{pk}_A, \text{pk}_B, \text{sk}_B)$
$\gamma \xleftarrow{\$} \mathbb{Z}_q$
$A \leftarrow h^{\gamma}$
A
$\delta \xleftarrow{\$} \mathbb{Z}_q$
$B \leftarrow h^{\delta}$
$\sigma_B \leftarrow \text{SoK-Sign}\left((A, B), \delta, \bar{S}_B, \text{sk}_B\right)$
B, σ_B
Abort if $\neg$ SoK-Verify $\left((A, B), \bar{S}_B, \sigma_B, \text{pk}_B\right)$
$\sigma_A \leftarrow \text{SoK-Sign}\left((A, B), \bar{S}_A, \gamma, \text{sk}_A\right)$
σ_A
Abort if $\neg$ SoK-Verify $\left((A, B), \bar{S}_A, \sigma_A, \text{pk}_A\right)$
$K \leftarrow B^{\gamma}$
$K \leftarrow A^{\delta}$
$k_{\text{sess}} \leftarrow \text{KDF}\,(A \,||\, B \,||\, \sigma_A \,||\, \sigma_B \,||\, K)$
$\rho \leftarrow \text{H}_q\,(k_{\text{sess}} \,||\, \text{"avow"})$
$\alpha \leftarrow \gamma + \rho$
$\beta \leftarrow \delta - \rho$
Shred γ, ρ
Shred δ, ρ

Figure 1: NOTRY-Kex. Alice and Bob follow the Key exchange protocol and store the masked secrets α, β for future avowal.

Second, Bob chooses his ephemeral secret $\delta \in \mathbb{Z}_q$ and computes $B \leftarrow h^{\delta}$. He generates the ADV proof π to the statement $\bar{S}_A$ defined in eq. (3.8) with the knowledge of ephemeral secret δ and his secret key sk_B. Based on π he generates a signature of knowledge, σ_B on message $m \leftarrow (A, B)$. Bob sends σ_B and B back to Alice.

Third, Alice verifies the σ_B from Bob with his public key pk_B and B. She aborts if σ_B is invalid. Alice computes her signature of knowledge σ_A on the statement $\bar{S_B}$ in eq. (3.9), with knowledge of her private key sk_A and γ on message $m \leftarrow (A, B, \sigma_B)$. In parallel, Alice is able to derive the DH key $K \leftarrow B^{\gamma}$. Alice can now generate the session key K_{sess} which is generated by a KDF that takes as input: the two SoKs σ_A, σ_B, the DH key K, and DH transcripts A, B. Finally, Alice masks her ephemeral secret with the digest ρ of the session key concatenated with the message “avow” to produce her avowal witness α. (Alice now erases γ and ρ). To finish NOTRY-Kex, Alice sends her SoK, σ_A, to Bob.

At last, Bob verifies σ_A from Alice and aborts if the verification fails. Next, Bob computes his DH key $K \leftarrow A^{\delta}$. Bob generates an identical session key to Alice, K_{sess}, following the same process. Bob gets his avowal witness β in such a way that $\alpha + \beta$ equals the addition of two ephemeral secrets $\gamma + \delta$. Finally, Bob securely deletes temporary secrets δ, ρ.

Note that the SoK statement in NOTRY-Kex is actually consistent with Equation 3.8. Take Alice as an example—she demonstrably does not know $dlog_h B$. Therefore, she is unable to prove both clauses involving B but to prove knowledge of her own ephemeral secret $\gamma \leftarrow dlog_{h}A$ and her private key $sk_A \leftarrow dlog_{g}pk_A$.

Forging key exchange transcripts. Using the simulator algorithm NOTRY-Kex-Sim, defined in Figure 3, anyone, including Alice and Bob, is able to generate indistinguishable NOTRY-Kex transcripts, which are A, B, σ_A, σ_B. The core idea of constructing the simulator is to generate $A, AB \leftarrow h^{\gamma}, g^{x}$ where γ, x is sampled from $\mathbb{Z}_q$. The simulator generates $B \leftarrow AB/A$. Therefore, the simulator is able to produce a valid ADV-based SoK by proving clauses $dlog_{g}AB \wedge dlog_{h}A$.

NOTRY-Kex-Ratchet. In most cases, a conversation between Alice and Bob consists of multiple messages. Therefore, a stream of keys is needed to secure a complete conversation consisting of a stream of messages between the two parties. To ensure future secrecy across an ongoing conversation we therefore introduce a ratcheted [59] version of our protocol, NOTRY-Kex-Ratchet. By sending a new SoK each time Alice and Bob update a key, we can extend NOTRY-Kex. As shown in Figure 2, NOTRY-Kex-Ratchet works as follows:

- For their first session key $K_{0,0}$, parties follow NOTRY-Kex as expected.
- For a message key $K_{i,j} (i > 0 \text{ } or \text{ } j > 0)$ indexed with i, j as i-th contribution from Alice and j-th contribution from Bob, first Alice samples a random ephemeral secret $\gamma_i \xleftarrow{\$} \mathbb{Z}_q$ and setups the DH key $K_{i,j} \leftarrow B_j^{\gamma_i}$. The message key $K_{sess_{i,j}}$ is the output of the KDF with $(A_i||B_j||\sigma_{A_i}||\sigma_{B_j}||K_{i,j})$ as input.

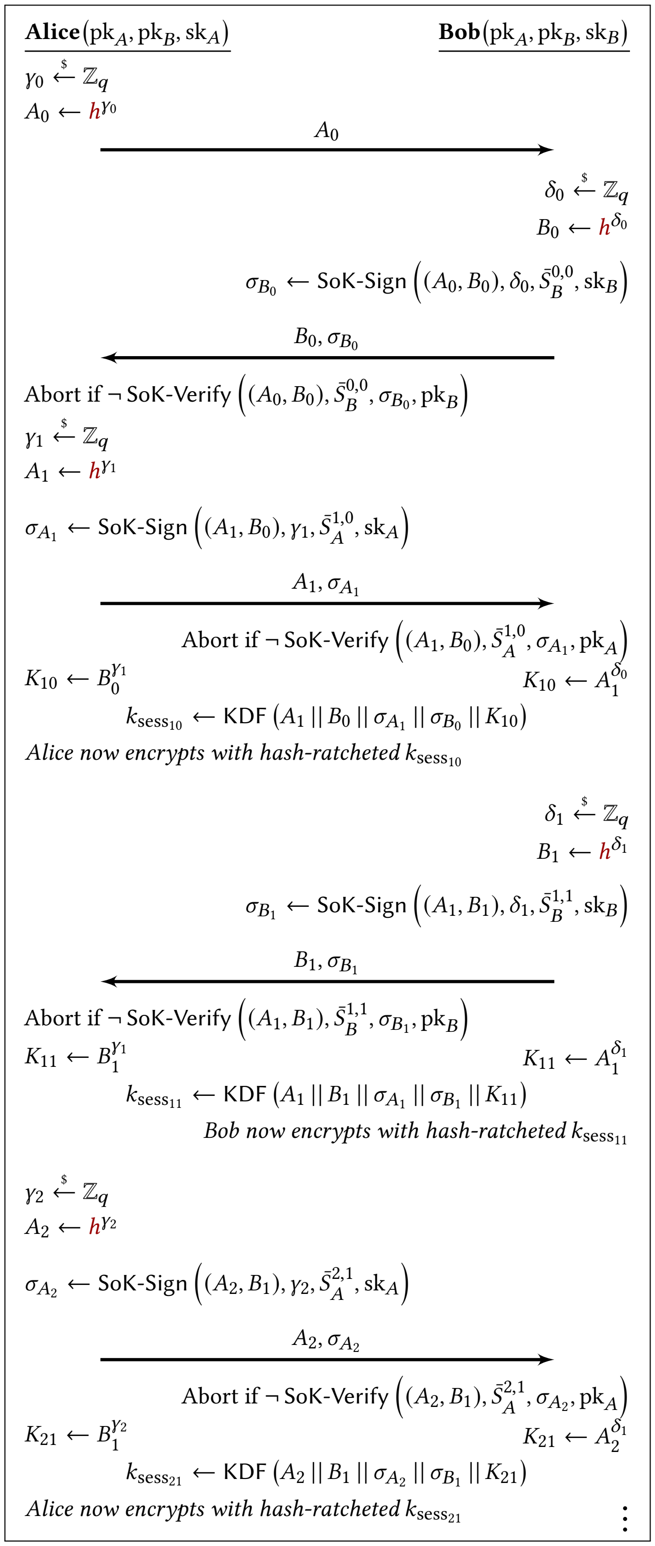
We instantiated the statement $S_A^{i,j}$ (and $S_B^{i,j}$ accordingly) with A_i, B_j, and pk_A (pk_B), where all of them are group elements.

4.2 NOTRY-Avow

Up to this point, our protocol allows Alice and Bob to establish a secure channel with NOTRY-Kex protocol and to deny a transcript using NOTRY-Kex-Sim. We now introduce the retroactive avowal routine within NOTRY protocol to support retroactive avowal. Recall that to avow, Alice and Bob aim to 'revoke' deniability. Deniability arises from the first AND clause in Equation 3.8 and 3.9, which proves knowledge of $\mathrm{dlog}_h B$ *or* $\mathrm{dlog}_h A$ and $\mathrm{dlog}_g(AB)$. The avowal procedure works by proving that the proof statement was generated through the second clause (involving the private key), i.e. by giving evidence that the first clause is false.

In a genuine conversation, Alice and Bob are incapable of proving knowledge of $\mathrm{dlog}_g AB$ but *can* prove knowledge of $\mathrm{dlog}_h(AB)$ — noting that this relies on the absence of a known relationship between g and h. The *Judge* can therefore be persuaded that the first clause is false by receiving $\mathrm{dlog}_h(AB)$. Note that Equation 3.8 and 3.9 both contain $\mathrm{dlog}_g AB$, proving non-knowledge of $\mathrm{dlog}_g AB$ indicating both statements are proven via their second clauses. That shows the two SoK signatures σ_A, σ_B were generated with knowledge of $\mathrm{sk}_A, \mathrm{sk}_B$.

Recall that two styles of avowal are possible, here we choose to give a construction for the designated verifier avowal where public avowal can be trivially derived by excluding the designated clause. Essentially, the designated verifier avowal proves non-knowledge of the discrete log to the designated verifier *Judge*. Therefore, the



$\mathbf{Alice}(\mathrm{pk}_A, \mathrm{pk}_B, \mathrm{sk}_A)$
$\mathbf{Bob}(\mathrm{pk}_A, \mathrm{pk}_B, \mathrm{sk}_B)$
$\gamma_0 \xleftarrow{\$} \mathbb{Z}_q$
$A_0 \leftarrow h^{\gamma_0}$
A_0
$\delta_0 \xleftarrow{\$} \mathbb{Z}_q$
$B_0 \leftarrow h^{\delta_0}$
$\sigma_{B_0} \leftarrow \text{SoK-Sign}\left((A_0, B_0), \delta_0, \bar{S}_B^{0,0}, \mathrm{sk}_B\right)$
B_0, σ_{B_0}
Abort if $\neg$ SoK-Verify $\left((A_0, B_0), \bar{S}_B^{0,0}, \sigma_{B_0}, \mathrm{pk}_B\right)$
$\gamma_1 \xleftarrow{\$} \mathbb{Z}_q$
$A_1 \leftarrow h^{\gamma_1}$
$\sigma_{A_1} \leftarrow \text{SoK-Sign}\left((A_1, B_0), \gamma_1, \bar{S}_A^{1,0}, \mathrm{sk}_A\right)$
A_1, σ_{A_1}
Abort if $\neg$ SoK-Verify $\left((A_1, B_0), \bar{S}_A^{1,0}, \sigma_{A_1}, \mathrm{pk}_A\right)$
$K_{10} \leftarrow B_0^{\gamma_1}$
$K_{10} \leftarrow A_1^{\delta_0}$
$k_{\mathrm{sess}_{10}} \leftarrow \text{KDF}\left(A_1 \,||\, B_0 \,||\, \sigma_{A_1} \,||\, \sigma_{B_0} \,||\, K_{10}\right)$
Alice now encrypts with hash-ratcheted $k_{\mathrm{sess}_{10}}$
$\delta_1 \xleftarrow{\$} \mathbb{Z}_q$
$B_1 \leftarrow h^{\delta_1}$
$\sigma_{B_1} \leftarrow \text{SoK-Sign}\left((A_1, B_1), \delta_1, \bar{S}_B^{1,1}, \mathrm{sk}_B\right)$
B_1, σ_{B_1}
Abort if $\neg$ SoK-Verify $\left((A_1, B_1), \bar{S}_B^{1,1}, \sigma_{B_1}, \mathrm{pk}_B\right)$
$K_{11} \leftarrow B_1^{\gamma_1}$
$K_{11} \leftarrow A_1^{\delta_1}$
$k_{\mathrm{sess}_{11}} \leftarrow \text{KDF}\left(A_1 \,||\, B_1 \,||\, \sigma_{A_1} \,||\, \sigma_{B_1} \,||\, K_{11}\right)$
Bob now encrypts with hash-ratcheted $k_{\mathrm{sess}_{11}}$
$\gamma_2 \xleftarrow{\$} \mathbb{Z}_q$
$A_2 \leftarrow h^{\gamma_2}$
$\sigma_{A_2} \leftarrow \text{SoK-Sign}\left((A_2, B_1), \gamma_2, \bar{S}_A^{2,1}, \mathrm{sk}_A\right)$
A_2, σ_{A_2}
Abort if $\neg$ SoK-Verify $\left((A_2, B_1), \bar{S}_A^{2,1}, \sigma_{A_2}, \mathrm{pk}_A\right)$
$K_{21} \leftarrow B_1^{\gamma_2}$
$K_{21} \leftarrow A_2^{\delta_1}$
$k_{\mathrm{sess}_{21}} \leftarrow \text{KDF}\left(A_2 \,||\, B_1 \,||\, \sigma_{A_2} \,||\, \sigma_{B_1} \,||\, K_{21}\right)$
Alice now encrypts with hash-ratcheted $k_{\mathrm{sess}_{21}}$
$\vdots$

Figure 2: NOTRY-Kex-Ratchet. To shred, Alice and Bob compute ρ, α, β as in Figure 1. For every racheted resulting prekey K_{xy}, we have $\mathrm{dlog}_{A_x} K_{xy} = \mathrm{dlog}_h B_y$ or $\mathrm{dlog}_{B_y} K_{xy} = \mathrm{dlog}_h A_x$.

```
Simulator
$\gamma, x \xleftarrow{\$} \mathbb{Z}_q$
$A \leftarrow h^{\gamma}$
$AB \leftarrow g^{x}$
$B \leftarrow AB/A$

$\sigma_A \leftarrow \textsf{SoK-Sign}\left((A, B), \gamma, \bar{S}_A, x\right)$        // Using $\gamma$, $x$

$\sigma_B \leftarrow \textsf{SoK-Sign}\left((A, B), \gamma, \bar{S}_B, x\right)$        // Using $\gamma$, $x$

$k_{\text{sess}} \leftarrow \textsf{KDF}\left(A \,||\, B \,||\, \sigma_A \,||\, \sigma_B \,||\, B^{\gamma}\right)$
$\rho \leftarrow \textsf{H}_q\left(k_{\text{sess}} \,||\, \text{“avow”}\right)$
$\alpha \leftarrow \gamma + \rho$
Shred $\gamma$, $\rho$
Output $A, B, \sigma_A, \sigma_B, \alpha, k_{\text{sess}}$
```

Figure 3: NOTRY-Kex-Sim. Anyone can run the simulator to generate an indistinguishable transcript to deny.

final avowal statement to be proven is:

$$[\text{dlog}_h(AB)] \vee [\textsf{Know}(sk_J)] \tag{4.1}$$

As the $\text{dlog}_g AB$ is separately held by Alice (α) and Bob (β), we, therefore, devise a two-party Σ proof generation by asking the two to generate their corresponding proof-of-knowledge of their secrets with Σ-OR Schnorr and exchange their proofs. The ultimate ADV avowal proof is an aggregation of the two proofs. Note that the statement P $(\neg P)$ in eq. (3.3) is instantiated with $\text{dlog}_g AB$ ($\text{dlog}_h AB$).

As shown in Figure 4, NOTRY-Avow works as follows:

(1) NOTRY-Avow begins by both parties executing NOTRY-Kex to get a session k_{rel} to communicate securely for the course of the avowal protocol. Note that a simpler deniable key exchange could also be used here to produce a session key for the avowal protocol, for simplicity, we assume we reuse NOTRY-Kex.
(2) Either Alice or Bob can request an avowal. Their counterparty approves this request by joining NOTRY-Avow. For convenience, we start NOTRY-Avow with Alice as the initiator. Alice samples random c_A, z_A, r_A for the ADV proof avowal, and samples another random value s_A to prevent message replay. Alice computes E_A as $g^{c_A} h^{z_A} \ell^{s_A}$ and $R_A \leftarrow h^{r_A}$ and sends them to Bob. Note that E_A is necessary to prohibit Alice from illegitimately generating a valid ADV proof that passes verification. We mitigate Alice's advantage (wherein she receives secrets first) by asking her to commit to all the secrets. Note that E_A is a natural adaptation of a Pedersen Commitment [58], where s_A is the random value to complete a Pedersen Commitment.
(3) Bob receives messages from Alice and randomly generates c_B, z_B, and r_B as his proof of the ADV proof avowal. Bob proceeds with getting $R_B \leftarrow h^{r_B}$ and sending it with encrypted c_B, z_B.

```
Alice$\left(\text{pk}_A, \text{pk}_B, \text{pk}_J, \text{sk}_A, \alpha\right)$                Bob$\left(\text{pk}_A, \text{pk}_B, \text{pk}_J, \text{sk}_B, \beta\right)$
            Execute NOTRY-Kex to produce $k_{\text{rel}}$,
            discarding the resulting release secrets.
$c_A \xleftarrow{\$} \{0,1\}^{\lambda}$
$z_A, s_A, r_A \xleftarrow{\$} \mathbb{Z}_q$
$E_A \leftarrow g^{c_A} h^{z_A} \ell^{s_A}$
$R_A \leftarrow h^{r_A}$
            $\xrightarrow{\quad E_A, R_A \quad}$
                                                        $c_B \xleftarrow{\$} \{0,1\}^{\lambda}$
                                                        $z_B, r_B \xleftarrow{\$} \mathbb{Z}_q$
                                                        $R_B \leftarrow h^{r_B}$
            $\xleftarrow{\quad R_B, \textsf{Enc}\left(k_{\text{rel}}, (c_B, z_B)\right) \quad}$
            $\xrightarrow{\quad \textsf{Enc}\left(k_{\text{rel}}, (c_A, z_A, s_A)\right) \quad}$
                                                        Abort if $E_A \neq g^{c_A} h^{z_A} \ell^{s_A}$
            $c_J \leftarrow c_A \oplus c_B$
            $z_J \leftarrow z_A + z_B$
            $R_J \leftarrow g^{z_J} / \text{pk}_J^{c_J}$
            $R_{AB} \leftarrow R_A R_B$
            $c \leftarrow \textsf{H}\left(R_{AB} \,||\, R_J \,||\, \cdots\right)$
            $c_{AB} \leftarrow c \oplus c_J$
$z_\alpha \leftarrow c_{AB}\alpha + r_A$                                $z_\beta \leftarrow c_{AB}\beta + r_B$
Output $z_\alpha$                                            Output $z_\beta$
            $z_{AB} \leftarrow z_\alpha + z_\beta$
            Output $\left(c_{AB}, z_{AB}, c_J, z_J\right)$
```

Figure 4: NOTRY-Avow

(4) Alice first decrypts the ciphertext to get Bob's partial proof and sends her partial proof c_A, z_A with fresh secret s_A secretly to Bob.
(5) After decrypting the ciphertext from Alice, Bob first checks message freshness by computing $E_A \overset{?}{=} g^{c_A} h^{z_A} \ell^{s_A}$. Observe that after three-more message exchanges both Alice and Bob will have acquired all of cs, zs, and Rs. Both Alice and Bob are then capable of generating the next stage of the avowal proof by setting $c_J \leftarrow c_A \oplus c_B$, $z_j \leftarrow z_A + z_B$, $R_{AB} \leftarrow R_A R_B$. After that, the simulated proof of sk_J is $R_J \leftarrow g^{z_J}/pk_J^{c_J}$. The Fiat-Shamir-based random challenge c of the disjunction proof is generated by hashing the message $R_{AB}||R_J$. The final challenge c_{AB}, required to prove $\text{dlog}_h AB$ (which is jointly generated Alice and Bob) is $c_{AB} \leftarrow c \oplus c_J$.
(6) Respectively, Alice and Bob generate the last piece of the avowal proof $z_w \leftarrow c_{AB} w + r_w$ with their partial witness $w \in [\alpha, \beta]$. They complete NOTRY-Avow by exchanging z_w to get $z_{AB} \leftarrow z_\alpha + z_\beta$ and output the avowal proof $\pi \leftarrow (c_{AB}, z_{AB}, c_J, z_j)$.

The designated *Judge* verifies π by checking whether it proves the knowledge $\text{dlog}_h AB$ or not. As shown in Figure 5, after parsing

```
$\mathbf{Judge}\left(\bar{\pi}', \left(\mathrm{dlog}_h AB \vee \mathrm{dlog}_g \mathrm{pk}_J\right)\right)$
$(c_{AB}, z_{AB}, c_J, z_J) \leftarrow \bar{\pi}'$
$R_{AB} \leftarrow h^{z_{AB}}/(AB)^{c_{AB}}$
$R_J \leftarrow g^{z_J}/\mathrm{pk}_J^{c_J}$
Accept if $c_{AB} \oplus c_J = \mathsf{H}\left(R_{AB} \,||\, R_J \,||\, \cdots\right)$, else Reject
```

Figure 5: NOTRY-Judge. Verifying an avowal proof $\bar{\pi}'$to the statement $(\mathrm{dlog}_h AB \vee \mathrm{dlog}_g \mathrm{pk}_J)$

the π from parties *Judge* recovers $R_{AB} \leftarrow h^{z_{AB}}/(AB)^{c_{AB}}$ and $R_J \leftarrow g^{z_J}/pk_J^{c_J}$. *Judge* declares a successful avowal if and only if $c_{AB} \oplus c_J = H(R_{AB}||R_J)$. If the formula does not hold, *Judge* announced that the avowal failed.

Avowing Multiple Messages. Alice and Bob may wish to simultaneously avow multiple messages, claiming ownership of an extended conversation. We can extend NOTRY-Avow to support this capability. Suppose Alice with $\Gamma \leftarrow [\gamma_1, \cdots, \gamma_n]$ and Bob with $\Delta \leftarrow [\delta, \cdots, \delta_n]$ intend to avow n messages. Parties roughly follow the original NOTRY-Avow protocol while integrating all avowal evidence $\Gamma(\Delta)$ into their partial proof z_w respectively. Specifically, $z_w \leftarrow c_{AB}\Gamma + r_w$ for Alice. As the *Judge* now receives aggregated avowal proofs π and $[AB_1, \cdots, AB_n]$ as inputs, we adjust the *Judge*'s verification procedure: *Judge* computes $R_{AB} \leftarrow h^{z_{AB}}/\prod_{i=0}^{n}(AB_i)^{c_{AB}}$. The rest of the verification follows as per NOTRY-Judge.

5 SECURITY

We will analyze three algorithms of NOTRY for our security proof. NOTRY-Kex is an authenticated key exchange (AKE) protocol. Parties in NOTRY-Kex, named Alice and Bob, after finishing NOTRY-Kex, generate a shared secret key and learn each other's identity. A secure AKE protocol ensures that they established the key with their intended partner and the key is a fresh secret. The key exchange protocol's transcript, $\mathcal{T}_{real}$, is the concatenation of all the messages exchanged between the two. The simulator algorithm NOTRY-Kex-Sim, depicted in Figure 3, produces a transcript $\mathcal{T}_{sim}$ without any secret keys to establish plausible deniability as anyone running the simulator could output a transcript $\mathcal{T}$ which is indistinguishable from the genuine transcript $\mathcal{T}$. For a UC-secure NOTRY, we are going to prove that continuous key agreement NOTRY-Kex-Ratchet preserves AKE and deniability under that UC framework.

NOTRY-Avow, our construction of MADV avowal proof needs to be a ZKPoK protocol. Second, for the interactive avowal generation, a UC-secure NOTRY requires a UC-secure NOTRY-Avow.

The remainder of this section is organized as follows: Section 5.1 overviews proof techniques that will be used for our proof, Section 5.2 defines desired security properties of NOTRY, and Section 5.3 demonstrates the functionality of NOTRY in our proof, Section 5.4 overviews our UC-secure proof of NOTRY. We defer the remainder of the proofs to appendices.

5.1 NOTRY Proof Overview

We proved the security of NOTRY with the Universally Composable Security (UC-secure) [16] framework. A UC-secure protocol is guaranteed to maintain security in an environment where different protocol instances run concurrently, even if the protocol is composed with arbitrary protocols.

Following [33, 66–68], we prove NOTRY is UC-secure under the external-subroutine universal composability framework (EUC) [33]. EUC is an extended generalized universal composability (GUC) framework [3, 17]. The GUC framework grants every entity in the model access to all the functionalities across composited protocol sessions. Therefore, it is regarded as a more convincing and accurate model of real-world settings. In addition, the GUC framework more naturally allows us to express the deniability property. GUC-based proofs can be simplified by constraining shared functionalities to a single common functionality $\mathcal{G}$, in the EUC model. Canetti et al. [17] proved that security under the EUC model is equivalent to proving it under the GUC model.

The key observation of our proof strategy is that NOTRY-Kex and NOTRY-Avow are not necessarily sequential. Parties can start avowal even if they haven't exchanged any keys. Of course, this attempt will either fail (generate an invalid proof) or be unconvincing (by not having any corresponding transcript to be avowed). Therefore, after defining ideal functionalities for NOTRY-Kex and NOTRY-Avow separately, proving NOTRY is secure requires proving NOTRY-Kex and NOTRY-Avow are UC-secure respectively. This special flavor of UC is called *multi-protocol UC*[15].

5.2 NOTRY Security Goals

NOTRY has the following security properties. We note that the first four are standard security notions for secure messaging and key exchange, whereas the fifth (mutually-agreed avowal) is novel to our setting:

(1) Universally composable AKE [22]: NOTRY-Kex-Ratchet realizes the ideal AKE functionality F_{kex} in the *Ideal World* in the UC framework [16]. UC-secure NOTRY-Kex-Ratchet enables the protocol to acquire arbitrary composability beyond the standard notions of an AKE protocol, which are mutual authentication, key privacy and freshness [7, 22].
(2) Deniability [30, 33, 39]: Transcripts in NOTRY-Kex can be efficiently forged with access only to the public parameters. Obviously, this deniability property demolishes the dependability of transcripts as proof of genuine key exchange protocol executions. A stronger notion of deniability, online deniability [33] guarantees that even a corrupted protocol participant (sometimes called an *informant*) cannot convince others of the authenticity of a transcript. NOTRY provides online deniability since NOTRY-Kex-Sim works without using Alice or Bob's long-term secret key.
(3) Future secrecy and post-compromise secrecy [6]: The exposure of long-term secret key(s) cannot undermine the confidentiality of past or future session key(s) respectively (following recovery by the protocol). We achieve future secrecy through ratcheting with NOTRY-Kex-Ratchet, in which both parties update their contribution to k_{sess} generation. The

```
NOTRY-Avow-Sim_Judge $\left((\text{dlog}_h AB \vee \text{dlog}_g \text{pk}_J), sk_J\right)$
$c_{AB} \xleftarrow{\$} \{0,1\}^\lambda$
$z_\alpha, z_{AB} \xleftarrow{\$} \mathbb{Z}_q$
$z_\beta \leftarrow z_{AB} - z_\alpha$
$R_{AB} \leftarrow h^{z_{AB}}/(AB)^{c_{AB}}$
$r_A \xleftarrow{\$} \mathbb{G}, R_A \leftarrow h^{r_A}$
$R_B \leftarrow R_{AB}/R_A$
$r_J \xleftarrow{\$} \mathbb{Z}_q$
$R_J \leftarrow g^{r_J}$
$c \leftarrow \mathsf{H}\left(R_{AB} \,||\, R_J \,||\, \cdots\right)$
$c_J \leftarrow c \oplus c_{AB}$
$z_J \leftarrow c_J \text{sk}_J + r_J$
$c_A \xleftarrow{\$} \{0,1\}^\lambda$
$c_B \leftarrow c_J \oplus c_A$
$z_A, s_A \xleftarrow{\$} \mathbb{Z}_q$
$z_B \leftarrow z_J - z_A$
$E_A \leftarrow g^{c_A} h^{z_A} \ell^{s_A}$
Execute NOTRY-Kex-Sim to produce $k_{\text{rel}}$.
Output $(c_{AB}, z_{AB}, c_J, z_J), E_A, R_A, R_B, k_{\text{rel}}$,
    NOTRY-Kex-Sim transcript,
    Enc $(k_{\text{rel}}, (c_A, z_A, s_A))$, Enc $(k_{\text{rel}}, (c_B, z_B))$.
```

Figure 6: NOTRY-Avow-Sim$_{\text{Judge}}$

secrecy of ratcheted key generation mechanisms is studied extensively by Alwen et al. [1] and Bienstock et al. [10].

(4) Post-specified peer [21]: Instead of specifying the identity of the intended peer when initiating NOTRY-Kex, parties learn the identity of their peers during protocol execution. The key observation is that Alice starts the protocol without involving any identity information about Bob. Bob's response is not related to pk_A either. Instead, Alice learns she was talking to Bob after she got the SoK of Bob.

(5) Mutually-agreed avowal: Transcripts based on session keys from NOTRY-Kex-Ratchet can be avowed by the two parties using NOTRY-Avow. Specifically, NOTRY-Avow is a two-party secure computation protocol that outputs a MADV avowal as a designated verifier (*Judge*) proof.

5.3 UC Security functionalities

In this section we outline our UC security functionalities, deferring full specifications to Appendix B. Our proof involves three ideal functionalities:

- F_{kex}, which runs key exchange in the absence of avowal. The two major functions of F_{kex} are to deliver shared session keys and to emulate multiple rounds of a continuous key agreement protocol.
- F_{avow}. For clarity and simplicity, we separate F_{avow} into two major functions, an ideal functionality for avowal proof generation and another one for avowal proof verification.
- $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$, which models miscellaneous key registeration with knowledge (*krk*) PKI and random oracles (*ro*). It's parameterized by a set of protocols interacting with $\mathbb{P}$, a number of random oracles n, a group $\mathbb{G}$ of order q with two independently sampled generators h, g. This is closely based on the global shared functionality defined in [68].

Ideal Functionality F_{kex} To prove NOTRY is UC-secure within the EUC-framework, we compile an ideal functionality F_{kex} to emulate NOTRY-Kex-Ratchet which captures all the security properties and features of this protocol. In addition to F_{kex}, shown in Algorithm 1, in the EUC-framework, there is an external environment **Z**, intended to distinguish between a simulator $\mathbb{S}$ in the *Ideal World* attacking the F_{kex} and a *Real World* adversary $\mathbb{A}$ attacking the real protocol. We denote dummy parties in the *Ideal World* as $\mathbb{P}$, the corresponding real parties in the *Real World* as $\bar{\mathbb{P}}$, and parties simulated by $\mathbb{S}$ for $\mathbb{A}$ as $\mathbb{P}^{\mathbb{S}}$.

Interactions between different entities in the context of EUC-framework are summarized as follows. First, **Z** is allowed to communicate with $\bar{\mathbb{P}}$ (or $\mathbb{P}$) and to write the inputs to every party $\mathbb{P}$ (in the *Ideal World*) or $\bar{\mathbb{P}}$ (in the *Real World*) and read their outputs. Likewise, $\mathbb{A}$ controls all the communication between every party $\bar{\mathbb{P}}$ (or $\mathbb{P}^{\mathbb{S}}$), whereas $\mathbb{S}$ is permitted to interact with F_{kex} under prescribed rules. Third, $\mathbb{S}$ and $\mathbb{A}$ are allowed to corrupt parties while **Z** will be immediately notified of this corruption. Finally, the ideal functionality F_{kex}, $\mathbb{S}$, $\mathbb{A}$, $\bar{\mathbb{P}}$, $\mathbb{P}$ are able to interact with global shared functionality $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$.

We model ratcheting by additionally introducing a round id *rid*. NOTRY-Kex-Ratchet is guaranteed to execute in rounds even in the general concurrent operation since NOTRY-Avow checks the consistency of *rid* before setting the shared key.

Ideal functionality F_{avow} For clarity and simplicity, taking advantage of the composability theorem, we defined a separate modular ideal functionality of avow F_{avow}. F_{avow} consists of two major functions: *Avow* and *Judge*. Note that, since *Judge* can be anyone to whom parties intend to avow, we therefore, consider *Judge* as part of F_{avow}. So the primary task to initialize F_{avow} is to register a Judge and then publish her public key. This models that parties know the identity of the *Judge* in the *Real World*. Naturally, as F_{avow} plays as *Judge*, it runs NOTRY-Avow-Sim$_{\text{Judge}}$ to produce transcripts for the simulator $\mathbb{S}$ in the *Ideal World*.

Ideal functionality $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$, depicted in Algorithm 4, is defined to model publicly accessible features/functions like PKI or CA. The Share functionality models three primary cross-session states. First is PKI since we assume that each party learns the other's long-term public key, as well as the key of the designated verifier judge pk_J. Second, a random oracle is needed for *KDF* functions, signatures-of-knowledge, avowal, and secret sharing in NOTRY-Kex. Essentially, $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$ combines the idea of $\mathcal{G}_{krk}$ *key registration of knowledge*, from Dodis et al. [33] to distribute public keys and reveal private keys to corresponding corrupted party. In addition, we also integrated the shared random oracle $\mathcal{G}_{ro}$ defined in Walfish [69] into $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$ for our proof working on EUC-framework.

The *non-information oracle* functionality is defined to ensure our ideal F_{kex} is relizeable. Since F_{kex} outputs a random key in the *Ideal World*, even classic DH key exchange does not securely realize F_{kex} under an adaptive attacker. $\mathbf{Z}$ can easily distinguish when a corrupted party's secret state is leaked to $\mathbf{Z}$ because there is no way for $\mathbb{S}$ to generate the counterpart's transcript to match the random output key and leaked secret. To handle adaptive adversarial corruptions, Canetti and Krawczyk [22] proposed a special *non-information oracle*. $\mathcal{N}$ as part of F_{kex}. $\mathcal{N}$ emulates a round of key exchange and exposes the secret state of the uncorrupted party to $\mathbb{S}$ once corruption happens. This helps $\mathbb{S}$ properly simulate a consistent key exchange under an adaptive adversary.

To capture online-deniability, Dodis et al. [33] observed that by relaxing the notion of fully adaptive adversaries to semi-adaptive adversaries we can achieve an online-deniable authentication given the PKI model. Semi-adaptive adversaries are restricted to corrupting a party only at the beginning of a protocol or at the end of it, but they can arbitrarily abort the protocol. After aborting, $\mathbb{S}$ gets the corresponding un-simulatable information from an *incriminate procedure*. This notion of deniability is known as *key exchange with incrimination abort* (KEIA), which assures deniability once the protocol is terminated by outputting a shared key. Note that if a protocol is executed without abort, it also implies perfect forward secrecy. This is because the session key is chosen randomly and it is independent of any information, including the protocol transcript.

5.4 UC-security proof sketch

Finally, we state our main security theorems and a proof sketch, deferring full details to Appendix C and Appendix D. We show a key lemma, that NOTRY-Avow is a ZKPoK scheme, in Section 5.4.1. Upon all the functionalities discussed above, we define UC-secure NOTRY-Kex and UC-secure NOTRY-Avow as following.

Theorem 2. *NOTRY-Kex-Ratchet EUC-realizes* F_{kex} *under the erasure* $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$*-hybrid model with adaptive security, given the DDH assumption and access to a non-information oracle* $\mathsf{NonInfo}_{NOTRY}$.

Theorem 3. *NOTRY-Avow EUC-realizes* F_{avow} *under the erasure* $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$*-hybrid model with adaptive security given the DDH assumption.*

Proof sketch To show that NOTRY-Kex-Ratchet actually EUC-realizes the F_{kex} is to show that for any PPT $\mathbb{A}$ attacking NOTRY in the *Real World*, there exists a corresponding PPT $\mathbb{S}$ in the *Ideal World* attacking F_{kex} such that the environment $\mathbf{Z}$ under the hybrid $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$ model is unable to distinguish whether it's in the *Real World* or *Ideal World* (Theorem 2, proof in Appendix C). The security proof (Appendix B) of Theorem 3 follows the same paradigm described above.

5.4.1 *NOTRY-Avow is a secure MADV avowal.*

We now present our proof that our NOTRY-Avow is a ZKPoK scheme, a key component of proving Theorem 3.

Lemma 1. NOTRY-Avow is a secure avowal of MADV proofs under Decisional Diffie–Hellman (DDH) in the random oracle model.

Remark 1. For simplicity, here we discuss single-message avowal rather than multi-message avowal. But our argument still holds in the latter case.

We divide proof of Lemma 1 into the following three lemmas:

Lemma 2. NOTRY-Avow MADV avowal proof is complete.

Proof. We argue the correctness of MADV avowal by examining the verification process. Since R_J is the simulator's output, *Judge* can simply recompute it again to get the identical value. For R_{AB}, first observe that $h^{z_{AB}}$ can be expanded to $h^{c_{AB}(\gamma+\sigma)+r_A+r_B}$ and $AB^{c_{AB}}$ equals to $h^{(\gamma+\sigma)c_{AB}}$. After dividing $h^{z_{AB}}$ with $AB^{c_{AB}}$, *Judge* gets $h^{r_A+r_B}$, which is identical to $R_A \times R_B$, for $R_A \leftarrow h^{r_A}, R_B \leftarrow h^{r_B}$. The random oracle will output an equivalent value as *Judge* and provers request with the same values. Therefore, verification will always pass for correctly generated proofs. □

Lemma 3. There exists a *PPT* extractor $\mathcal{E}$ for NOTRY-Avow MADV avowal can extract a valid witness while $\mathbb{A}$ is unable to get the witness from a genuine ADV proof $\bar{\pi}$.

Proof. We construct a knowledge extractor to demonstrate how to get secrets α, β with the forking lemma[60]. Note that $\mathcal{E}$ works the same way to extract both secrets. Suppose that $\mathcal{E}$ can rewind a verifier in the NOTRY-Avow to have the challenge c in two proofs. The extractor $\mathcal{E}$ will run NOTRY-Avow twice to extract α. In the first round, upon observing that Alice outputs her partial proof z_α, the $\mathcal{E}$ records the partial proof z_β and rewinds Bob to compel him to output the same c. In this case, Alice will generate two proofs for the same challenge c. With two partial proofs $z_\alpha, \bar{z_\alpha}$ received from Alice, to get secret α, $\mathcal{E}$ first collects two common challenge values running NOTRY-Avow $c_{AB}, \overline{c_{AB}}$. Now $\mathcal{E}$ can derive α by dividing $z_\alpha - \bar{z_\alpha}$ with $c_{AB} - \overline{c_{AB}}$. Observe that $z_\alpha - \bar{z_\alpha} = (c_{AB}\alpha + r_A) - (\overline{c_{AB}}\alpha + r_A)$ equals to $c_{AB}\alpha - \overline{c_{AB}}\alpha$. So $\alpha \leftarrow (z_\alpha - \bar{z_\alpha})/(c_{AB} - \overline{c_{AB}})$.

Avowal soundness can be trivially satisfied for NOTRY-Avow. Even though the witness of avowal proof serves as a partial witness to the ADV proof, $\mathbb{A}$ learns nothing about the ADV witness from a zero-knowledge ADV proof $\bar{\pi}$. □

Remark 2. To illustrate the idea, we follow the classical way of constructing the extractor $\mathcal{E}$ with the rewinding technique. However, since rewinding is impermissible in the UC model, *straight-line compilers*, which are extensively studied in the literature[37, 40, 45, 54, 62], can be directly applied to NOTRY-Avow to transform this special Σ-OR proof to a UC-secure one.

Remark 3. A global random oracle is not enough to enable a straight-line compiler in the UC model[18]. To extract the witness, a knowledge extractor $\mathcal{E}$ without rewinding gets a special power: *oberserving* all requests to the random oracle and their responses. We, therefore, adopt *restricted observable global random oracles*[19]. Also, note that since we apply the Fiat-Shamir transform to get a non-interactive protocol, this special flavor of *random oracle* is also needed for *special-soundness* for both $\bar{\pi}$ and $\bar{\pi}'$.

Lemma 4. NOTRY-Avow MADV avowal is zero-knowledge under DDH/RO and a semantic-secure encryption scheme.

Proof. Since it is a designated verifier proof, *Judge* can easily simulate a correct transcript. Note that this doesn't damage generalization because anyone can serve as a *Judge*. Therefore, we can construct a simulator NOTRY-Avow-$\mathsf{Sim}_{\mathsf{Judge}}$ (abbreviate it to

$\mathcal{S}_{Judge}$), shown in Figure 6, that outputs indistinguishable transcripts of NOTRY-Avow. The information that the simulator takes advantage of is AB and pk_J. Basically, the idea used to construct $\mathcal{S}_{Judge}$ is to prove the second clause of the statement Equation (4.1) and then generate the proof. First, $\mathcal{S}_{Judge}$ picks c_{AB} and samples z_A, z_{AB} from $\mathbb{Z}_q$. Set $z_\beta \leftarrow z_{AB} - z_A$, $R_{AB} \leftarrow h^{z_{AB}}/(AB)^{c_{AB}}$. Select a random R_A and set $R_B \leftarrow R_{AB}/R_A$. In this way, R_{AB} will always be coherent to the verification of NOTRY-Judge. Second, $\mathcal{S}_{Judge}$ moves to prove knowledge of sk_J. Getting a random pair (r_J, R_J) for computing the Fiat-Shamir challenge value c. Set $c_J \leftarrow c \oplus c_{AB}$ and the response value $z_J \leftarrow c_J \mathsf{sk}_J + r_J$. Third, $\mathcal{S}_{Judge}$ secretly shares c_J and z_J to simulate the way they are generated in NOTRY-Avow. Finally, after simulating a correct ADV proof, $\mathcal{S}_{Judge}$ emulates the rest of the communication by running NOTRY-Kex-Sim, selecting a random s_A to set up E_A, and compiling two ciphertexts $(\bar{c_1}, \bar{c_2}) \leftarrow (\mathsf{Enc}(k_{\mathrm{rel}}, (c_A, z_A, s_A)), \mathsf{Enc}(k_{\mathrm{rel}}, (c_B, z_B)))$.

It's trivial to test the correctness of the simulated proof. As $g^{z_J} = g^{c_J \mathsf{sk}_J + r_J}$, when it divides $\mathsf{pk}_J^{c_J}$ we get g^{r_J}, which matches the check.

To show indistinguishability between the simulated transcript and the original transcript, we first observe the output of the simulator is $(\bar{\pi}, E_A, R_A, R_B, \bar{c_1}, \bar{c_2})$. The partial simulated proof $(c_J, z_{AB}) \in \bar{\pi}$ is randomly sampled, while $z_J \in \bar{\pi}$ is masked by random values r_J, and R_J is shadowed by a random oracle output c and random value c_{AB}. Thus, as each proof element of $\bar{\pi} = (z_J, z_{AB}, c_J, c_{AB})$ is either random or masked with a random value, we conclude that $\bar{\pi}$ is indistinguishable from random. The same argument can also be applied to prove that π is also indistinguishable from random. Therefore a simulated proof $\bar{\pi}$ is indistinguishable from π.

Two ciphertexts $\bar{c_1}, \bar{c_2}$ are indistinguishable from c_1, c_2 in NOTRY-Avow since the encryption scheme is semantic secure. The E_A, R_A both are generated from random values while R_B is masked by R_A, which implies all three values share the identical distribution to their counterparts in NOTRY-Avow. □

6 REAL WORLD EVALUATION

We designed our protocol for use in real-world systems and accordingly developed a proof-of-concept implementation, which we evaluate here. An anonymized review-ready copy of our code is available at https://github.com/xxsqwe/notry.

Our NOTRY implementation is based on the official Rust implementation of Signal[53] and is designed to achieve an equivalent security level (128 bits). In particular, we use the same Curve 25519 [9] as Signal. Our implementation extends the Signal implementation to support both single- and multiple-message transcript avowal. All experiments were performed on an Intel 12th generation core i7-12700K pinned to 3.6GHz with 32GB RAM.

Our goal in evaluation is to determine the performance penalty incurred by implementing NOTRY over a non-avowable secure messaging protocol. We, therefore, compare it against Signal, which represents the best-in-class for secure messaging with deniability. Additional overhead is paid for a UC-secure Signal [10]. Therefore, we implement NOTRY without a *straight-line* compiler for a fair comparison.

6.1 Results

First, we profile the computation and communication overhead in the key exchange stage. This gives an indication of the overhead that NOTRY would exhibit in regular use.

As session keys in Signal are generated by the X3DH protocol, which employs double ratcheting with its accompanying performance costs, we implement a similarly ratcheted NOTRY key exchange. We collected our performance results from 10,000 experimental evaluations on our NOTRY-Kex implementation and Signal benchmarks, summarized in Table 1. While NOTRY incurs an approximately 4× communication and 8× computational overhead as compared with the Signal protocol, the absolute time taken per key exchange (~ 1ms) is still negligible — demonstrating that the protocol may be sufficiently performant for eventual use in production systems.

We also evaluate the cost of performing avowal, both for the communicating parties and for the *Judge*. Our protocol is designed to allow any server to play the role of a *Judge* and the cost of verification is therefore not a bottleneck for the protocol.

We measured the cost of running the avowal with 1, 10, 100, and 1000 transcripts respectively, and tabulate our results in Figure 7. We find that the communication and performance costs to the communicating parties are roughly constant, and scales linearly for the *Judge*. While it is unsurprising that the computational burden falls to the *Judge* the overall performance costs are both small and linear in the number of transcripts to avow, and therefore pose a lesser deployment concern.

Table 1: NOTRY and Signal performance evaluation for key exchange

	Signal	NOTRY	Ratio
Key Generation [ms]	0.011±0.001	0.044±0.002	4×
SoK Generations[ms]	-	0.174±0.01	N/A
SoK Verify [ms]	-	0.230±0.01	N/A
Key Exchange [ms]	0.151±0.03	1.21±0.05	8×
Rounds	2	2	1×
Public key [Bytes]	32	32	1×
Prekey [Bytes]	128	-	N/A
SoK [Bytes]	-	256	N/A
Key Exchange [B/key]	80	335	4×

7 DISCUSSION AND CONCLUSIONS

We identified an interesting potential weakness with existing deniable messaging applications: sometimes both participants in a conversation initially meant to be deniable may genuinely wish to remove deniability and *avow* the conversation at a later time. To overcome this limitation, we developed a novel notion, ADV proofs, and applied a practical ADV proof construction to construct NOTRY. To the best of our knowledge, this is the first DAKEA protocol that retains all the critical security properties of DAKE protocols. Our evaluation shows that NOTRY protocol is also a reasonably efficient DAKEA protocol in practice.

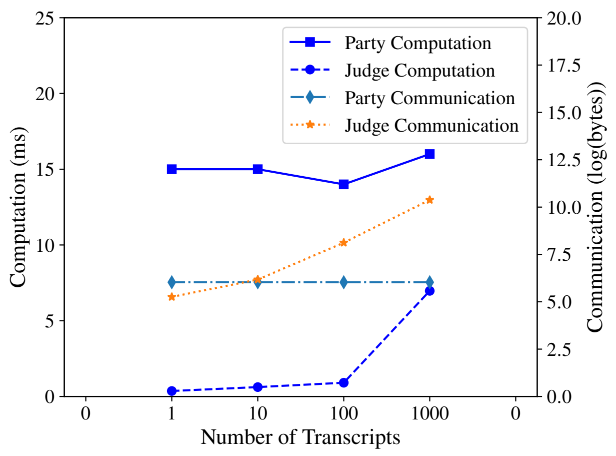


Figure 7: Avowal Overhead of a Party and the Judge

Some of the techniques in our work may be of interest to other application areas. In particular, our construction of avowable designated verifier proofs may have other applications for systems aiming to provide some flavor of revocable deniability. There also may be other constructions possible for ADV proofs with efficiency advantages in some settings, such as the hash-based constructions we suggested. Constructing efficient ADV proofs under different security assumptions is an interesting research direction.

We would especially like to see the following work that constructs an efficient DAKEA protocol that is quantum-resistant. More generally, this intuition might be extended or equivalent to a new zero-knowledge paradigm, that is, can we prove not owning the witness to an NP statement by proving knowing a witness to a polynomial-time reduction to another statement? The two statements are inverse to each other.

Finally, our work raises interesting questions for deniable messaging. The real-world value of deniability remains an open question, leaving the value of revocable deniability in question as well. There are also many open questions about building NOTRY into a practical system, in particular with handling revocation secrets and providing an acceptable user interface for avowal.

ACKNOWLEDGMENTS

We thank the anonymous reviewers of CCS 23 and PETS 23 for valuable suggestions. We also thank our PETs Shepherd for helpful feedbacks. Faxing Wang is supported by Melbourne Research Scholarship. Riad Wahby and Joseph Bonneau are supported by DARPA under Agreement No. HR00112020022. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government or DARPA.

REFERENCES

[1] J. Alwen, S. Coretti, and Y. Dodis. The double ratchet: security notions, proofs, and modularization for the signal protocol. In *Asiacrypt*, 2019 (cited on pages 391, 400).

[2] J. Alwen, S. Coretti, Y. Dodis, and Y. Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In *CRYPTO*, 2020 (cited on page 391).

[3] C. Badertscher, R. Canetti, J. Hesse, B. Tackmann, and V. Zikas. Universal composition with global subroutines: capturing global setup within plain uc. In *TCC*, 2020 (cited on page 399).

[4] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-12, Internet Engineering Task Force, 2022. URL: https://datatracker.ietf.org/doc/draft-ietf-mls-protocol/. Work in Progress (cited on page 391).

[5] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *CRYPTO*, 2001 (cited on page 391).

[6] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Eurocrypt*, 2000 (cited on pages 393, 399).

[7] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *CRYPTO*, 1994 (cited on pages 393, 399).

[8] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *ACM STOC*, 1995 (cited on page 393).

[9] D. J. Bernstein. Curve25519: new diffie-hellman speed records. In *PKC*, 2006 (cited on page 402).

[10] A. Bienstock, J. Fairoze, S. Garg, P. Mukherjee, and S. Raghuraman. A more complete analysis of the signal double ratchet algorithm. In *IACR CRYPTO*, 2022 (cited on pages 400, 402).

[11] D. Boneh and C. Komlo. Threshold signatures with private accountability. In *CRYPTO*, 2022 (cited on page 393).

[12] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In *ACM WPES*, 2004 (cited on pages 391, 393, 396).

[13] C. Boyd, W. Mao, and K. G. Paterson. Key agreement using statically keyed authenticators. In *ACNS*, 2004 (cited on page 393).

[14] C. Brzuska, M. Fischlin, B. Warinschi, and S. C. Williams. Composability of bellare-rogaway key exchange protocols. In *ACM CCS*, 2011 (cited on page 393).

[15] J. Camenisch, M. Drijvers, and B. Tackmann. Multi-protocol uc and its use for building modular and efficient protocols. Cryptology ePrint Archive, Paper 2019/065, 2019 (cited on page 399).

[16] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *IEEE FOCS*, 2001 (cited on pages 392, 399).

[17] R. Canetti, Y. Dodis, R. P. Pass, and S. W. Walfish. Universally composable security with global setup. In *TCC*, 2006 (cited on page 399).

[18] R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO*, 2001 (cited on page 401).

[19] R. Canetti, A. Jain, and A. Scafuro. Practical uc security with a global random oracle. In *ACM CCS*, 2014 (cited on page 401).

[20] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Eurocrypt*, 2001 (cited on page 393).

[21] R. Canetti and H. Krawczyk. Security analysis of IKE's signature -based key-exchange protocol. In *CRYPTO*, 2002 (cited on page 400).

[22] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *Eurocrypt*, 2002 (cited on pages 399, 401, 407).

[23] M. Chase and A. Lysyanskaya. On signatures of knowledge. In *CRYPTO*, 2006 (cited on page 395).

[24] D. Chaum. Zero-knowledge undeniable signatures. In *Eurocrypt*, 1990 (cited on page 392).

[25] D. Chaum and H. V. Antwerpen. Undeniable Signatures. In *CRYPTO*, 1989 (cited on page 393).

[26] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO*, 1994 (cited on page 394).

[27] C. Cremers and M. Feltz. Beyond eck: perfect forward secrecy under actor compromise and ephemeral-key reveal. In *ESORICS*, 2012 (cited on page 393).

[28] I. Damgård. On sigma protocol. https://www.cs.au.dk/~ivan/Sigma.pdf (cited on page 406).

[29] C. D. de Saint Guilhem, M. Fischlin, and B. Warinschi. Authentication in Key-Exchange: Definitions, Relations and Composition. In *IEEE CSF*, 2020 (cited on page 393).

[30] M. Di Raimondo, R. Gennaro, and H. Krawczyk. Deniable authentication and key exchange. In *ACM CCS*, 2006 (cited on pages 393, 399).

[31] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 1992 (cited on page 393).

[32] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 1976 (cited on page 393).

[33] Y. Dodis, J. Katz, A. Smith, and S. Walfish. Composability and on-line deniability of authentication. In *TCC*, 2009 (cited on pages 392, 393, 399–401).

[34] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *ACM STOC*, 1991 (cited on page 393).

[35] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. *JACM*, 2004 (cited on page 393).

[36] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 1985 (cited on page 391).

[37] M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*, 2005 (cited on page 401).

[38] M. Fischlin, F. Günther, B. Schmidt, and B. Warinschi. Key confirmation in key exchange: a formal treatment and implications for tls 1.3. In *IEEE Security & Privacy*, 2016 (cited on page 393).

[39] M. Fischlin and S. Mazaheri. Notions of deniable message authentication. In *ACM WPES*, 2015 (cited on page 399).

[40] C. Ganesh, Y. Kondi, C. Orlandi, M. Pancholi, A. Takahashi, and D. Tschudi. Witness-succinct universally-composable snarks. In *Eurocrypt*, 2023 (cited on page 401).

[41] P. Grubbs, J. Lu, and T. Ristenpart. Message Franking via Committing Authenticated Encryption. In *CRYPTO*, 2017 (cited on page 392).

[42] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated Verifier Proofs and their Applications. In *Eurocrypt*, 1996 (cited on page 393).

[43] S. Jiang and R. Safavi-Naini. An efficient deniable key exchange protocol. In *FC*, 2008 (cited on page 393).

[44] J. Katz. Efficient and non-malleable proofs of plaintext knowledge and applications. In *Eurocrypt*, 2003 (cited on page 393).

[45] Y. Kondi and A. Shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. In *Asiacrypt*, 2022 (cited on page 401).

[46] H. Krawczyk. Hmqv: a high-performance secure diffie-hellman protocol. In *CRYPTO*, 2005 (cited on page 393).

[47] H. Krawczyk. Sigma: the 'sign-and-mac'approach to authenticated diffie-hellman and its use in the ike protocols. In *CRYPTO*, 2003 (cited on page 393).

[48] H. Krawczyk. Skeme: a versatile secure key exchange mechanism for internet. In *NDSS*, 1996 (cited on page 393).

[49] H. Krawczyk and T. Rabin. Chameleon Signatures. In *NDSS*, 2000 (cited on page 393).

[50] B. LaMacchia, K. Lauter, and A. Mityagin. Stro-nger security of authenticated key exchange. In *ProvSec*, 2007 (cited on page 393).

[51] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes, and Cryptography*, 2003 (cited on page 393).

[52] Y. Li and S. Schäge. No-match attacks and robust partnering definitions: defining trivial attacks for security protocols is not trivial. In *ACM CCS*, 2017 (cited on page 393).

[53] libsignal. Signal Foundation. https://github.com/signalapp/libsignal (cited on page 402).

[54] A. Lysyanskaya and L. N. Rosenbloom. Universally composable sigma-protocols in the global random-oracle model. In *TCC*, 2022 (cited on page 401).

[55] M. Marlinspike and T. Perrin. The X3DH Key Agreement Protocol. Signal Foundation, 2016. https://signal.org/docs/specifications/x3dh/x3dh.pdf (cited on pages 391–393).

[56] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA*, 2001 (cited on page 391).

[57] S. Park and A. Sealfon. It Wasn't Me! Repudiability and Claimability of Ring Signatures. In *CRYPTO*, 2019 (cited on page 393).

[58] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991 (cited on page 398).

[59] T. Perrin and M. Marlinspike. The double ratchet algorithm. Signal Foundation, 2016. https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf (cited on pages 392, 397).

[60] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*:361–396, 2000 (cited on pages 401, 406).

[61] M. O. Rabin. How to exchange secrets with oblivious transfer. *CRYPTOl. ePrint Arch.*, 1985 (cited on page 391).

[62] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Annual Symposium on Foundations of Computer Science FOCS*, 1999 (cited on page 401).

[63] S. Schäge. Topas: 2-pass key exchange with full perfect forward secrecy and optimal communication complexity. In *ACM CCS*, 2015 (cited on page 393).

[64] C.-P. Schnorr. Efficient signature generation by smart cards. In *CRYPTO*, 1989 (cited on pages 394, 396, 405).

[65] N. Tyagi, P. Grubbs, J. Len, I. Miers, and T. Ristenpart. Asymmetric Message Franking: Content Moderation for Metadata-Private End-to-End Encryption. In *CRYPTO*, 2019 (cited on page 392).

[66] N. Unger. *End-to-End Encrypted Group Messaging with Insider Security*. PhD thesis, University of Waterloo, 2021 (cited on pages 392, 393, 399).

[67] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith. Sok: secure messaging. In *IEEE Security & Privacy*, 2015 (cited on page 399).

[68] N. Unger and I. Goldberg. Improved strongly deniable authenticated key exchanges for secure messaging. In *PETS*, volume 2018 of number 1, 2018 (cited on pages 392, 399, 400).

[69] S. Walfish. *Enhanced Security Models for Network Protocols*. PhD thesis, New York University, 2008 (cited on pages 393, 400).

[70] A. C.-C. Yao and Y. Zhao. Oake: a new family of implicitly authenticated diffie-hellman protocols. In *ACM CCS*, 2013 (cited on page 393).

[71] T.-Y. Youn, C. Lee, and Y.-H. Park. An efficient non-interactive deniable authentication scheme based on trapdoor commitment schemes. *Computer Communications*, 2011 (cited on page 393).

A MADV PROOFS $\bar{\pi}$

A.1 Secure MADV proofs $\bar{\pi}$

Definition 4. A secure MADV proof $\bar{\pi}$ consists of a triple of algorithms (Setup, Avow, Judge) with the following properties:

(1) *Completeness:* Any prover who generated an MADV proof $\bar{\pi}$ with knowledge of $\bar{W}$ satisfies:

$$\Pr\left[1 \leftarrow Verify(\bar{\pi}, \bar{S}) \;\middle|\; \bar{\pi} \leftarrow Gen(\bar{S}, \bar{W})\right] = 1 \tag{A.1}$$

Completeness indicates that *Judge* always accepts the MADV avowal proof $\bar{\pi}'$ from an honest prover.

(2) *Special Knowledge Soundness*: There exists a *PPT* extractor $\mathcal{E}$ which, given two transcripts of avowal proof $\mathcal{T}_1$ and $\mathcal{T}_2$, can extract a valid witness:

$$\Pr\left[1 \leftarrow Verify(\bar{\pi}, P) \;\middle|\; \begin{array}{l} \bar{W}' \leftarrow \mathcal{E}(1^\lambda, \mathcal{T}_1, \mathcal{T}_2) \\ \bar{\pi} \leftarrow Gen(\bar{S}, \bar{W}') \end{array}\right] = 1 - \mathsf{negl}(\lambda) \tag{A.2}$$

where W_P' is the extracted witness. Special soundness implies that any party who generates a valid ADV proof $\bar{\pi}$ must gain the knowledge of the witness W_P to the statement P.

(3) *Honest Verifier Zero-Knowledge*: There exists a *PPT* simulator $\mathbb{S}$ that generates an MADV proof $\bar{\pi}_{\mathbb{S}}$ without knowledge of the witness w such that $1 \leftarrow Judge(\bar{\pi}_{\mathbb{S}})$. $\bar{\pi}_{\mathbb{S}}$ holds that:

$$\bar{\pi}_{\mathbb{S}} \approx_c \bar{\pi} \tag{A.3}$$

A.2 Construction of MADV proofs

Here, we detail the proof generation and verification protocol. Note that MADV proofs are designed under the context of *Signature of Knowledge* in NOTRY-Kex, so we instantiate the Signature of Knowledge with the Schnorr identification scheme and the Schnorr signature scheme.

Classical Schnorr Identification Scheme: [64] for example, targeting on proving knowledge of $\gamma \leftarrow \mathsf{dlog}_h A$, given global parameters $\mathbb{G}, q, g, h$, Alice first samples $r \xleftarrow{\$} \mathbb{Z}_q$ and computes $a = g^r \ mod\ P$. Next, she applies the Fiat-Shamir transform to get a non-interactive proof, setting $c = H(a||A||h)$. Last, she sends the proof $(a, z = r + c\gamma)$ to Bob. Bob verifies the proof by first reconstructing the random challenge $c' = H(a||A||h)$, then checking if $h^z \overset{?}{=} a \cdot A^{c'}$.

Remark 4. Challenge c will be generated in another way when applying a *straight-line compiler* to bootstrap the protocol to a UC-secure one. Briefly, given a hash function H_{sl}, Bob only accepts a proof iff the $H_{sl}(a||c||z||A||h)$ starts with x bits long leading zeros and passes verification. In this case, given a commitment a, Alice is supposed to try multiple potential challenges until she gets a valid challenge c. Also, since H_{sl} outputs x long digest, where $x \cdot y = \lambda$, so to achieve the λ-bit level of security, Alice has to generate y proof transcripts $\{(a_i, z_i)\}_{i \in [y]}$ and ensure that the corresponding hash of every proof $H_{sl}(\mathbf{a}||c_i||z_i||A||h)$ has x leading zeros, where $\mathbf{a} = (a_1, \cdots, a_i)$. Alice wraps (a, c, z) up as the Schnorr proof under the *straight-line compiler*.

Schnorr AND Proof: proving an AND clause via Schnorr protocol is to generate two proofs for the two clauses respectively. For example, to prove $\mathsf{dlog}_h A \wedge \mathsf{dlog}_g pk_A$, Alice generates the Schnorr proof to $\mathsf{dlog}_h A$ as we discussed before. Second, in the same way, Alice generates the Schnorr proof $(\bar{a}, \bar{z})$ to $\text{dlog}_g pk_A$. Alice sends two proofs $((a,z),(\bar{a},\bar{z}))$ to Bob who verifies two proofs individually as the basic scheme.

Schnorr OR Proof: [28] for the OR clause, i.e. $\text{dlog}_h A \vee \text{dlog}_h B$, since Alice has non-knowledge of $\text{dlog}_h B$, she will simulate a proof for $\text{dlog}_h B$. To get a simulated proof, Alice first samples $c_B, z_B \overset{\$}{\leftarrow} \mathbb{Z}_q$, then she finalizes simulation with $a_B \leftarrow g^{z_B}/B^{c_B}$. For the proof to $\text{dlog}_h A$, Alice generates r_A, a_A as the basic Schnorr scheme. Second, she gets the challenge value $c \leftarrow H(a_A||a_B||A||B||h)$, computes her really challenge $c_A \leftarrow c \oplus c_B$ and completes the proof with $z_A \leftarrow r + c_A\gamma$ as the basic scheme. Alice sends two proofs (r_A, z_A),(r_B, z_B) to Bob. The verifier Bob first reconstructs $a'_A \leftarrow g^{z_A}/A^{c_A}, a'_B \leftarrow g^{z_B}/B^{c_B}$. Bob accepted a proof if $c_A \oplus c_B = H(a'_A||a'_B||A||B||h)$, otherwise rejects it.

A.3 MADV proofs in NOTRY are ZKPoK

Lemma 5. The instantiation of the MADV proof in NOTRY-Kex is *complete*, *special sound*, and *zero-knowledge* under DDH/RO assumption.

Remark 5. Every security property of an SoK scheme is captured by its corresponding property of MADV proofs defined in Appendix A. *Correctness* is defined via *Completeness*, *Simulatability* is defined by *Zero-knowledge*, and *Extraction* is defined via *Special Knowledge Soundness.*

Proof. *Completeness*: The MADV proof in NOTRY-Kex consists of four individual Schnorr proofs with respect to four clauses of the $\bar{S}_A$ in Equation (3.8). Specifically, Alice generates the MADV proof (also signature of knowledge) $\bar{\pi}$ with $\gamma \leftarrow \text{dlog}_h A$ and her private key sk_A where

$$\bar{\pi} = \{\underbrace{(a_A, c_A, z_A)}_{\bar{\pi}_A}, \underbrace{(a_B, c_B, z_B)}_{\bar{\pi}_B}, \underbrace{(a_J, c_J, z_J)}_{\bar{\pi}_J}, \underbrace{(a_{AB}, c_{AB}, z_{AB})}_{\bar{\pi}_{AB}}\} \quad (A.4)$$

$\bar{\pi}_B$ and $\bar{\pi}_{AB}$ are simulated proofs. It's trivial to tell that all four proofs will pass verification as each proof is a Schnorr proof.

Special soundness: Two accepting proofs with the same first commitment are enough to retrieve the knowledge under the Schnorr protocol. With rewinding, the knowledge extractor $\mathcal{E}$ works similarly as the $\mathcal{E}$ in NOTRY-Avow. The only distinction in Schnorr-OR proofs is that $\mathcal{E}$ is supposed to output an additional bit to indicate which clause was proved. Taking $\text{dlog}_h A \vee \text{dlog}_h B$ as an example, $\mathcal{E}$ rewinds Alice to the point where she asks the random oracle to generate two accepting proofs: $(a_A, c_A, z_A), (a_B, c_B, z_B)$ and $(a_A, c'_A, z'_A), (a_B, c'_B, z'_B)$. If $c_A \neq c'_A$, $\mathcal{E}$ outputs 0 to indicate that $\text{dlog}_h A$ is proved and derives γ from $\bar{\pi}_A, \bar{\pi}'_A$ the same way as the $\mathcal{E}$ in Lemma 3.

On the other hand, consider applying the *straight-line compiler* to MADV avowal proofs in Remark 4, $\mathcal{E}$ will get to search all the queries that Alice made to the random oracle. Since rewinding is not allowed in the UC model. *observerable random oracle*, defined in Algorithm 4, exposes an additional interface *observer-RO* to $\mathcal{E}$ for extracting the witness. After receiving $\{\bar{\pi}_i\}_{i\in[y]}$ from Alice, $\mathcal{E}$ first parses $(a_i, c_i, z_i)_{i\in[y]} \leftarrow \bar{\pi}$ and sets $\mathbf{a} = (a_i)_{i\in[y]}$. Second, $\mathcal{E}$ sends *observe-RO*(Alice) to the random oracle and gets back all the requests Alice made. Finally, $\mathcal{E}$ finds two requests i, j that share the $\mathbf{a}$ but $c_j \neq c_i$ and $z_j \neq z_i$ and pass Schnorr-OR verification. By the forking lemma[60], $\mathcal{E}$ extracts the witness γ from $(z_j - z_i)/(c_j - c_i)$.

Honest verifier zero-knowledge: we give a simulator $\mathbb{S}_{OR}$ that simulates indistingsuiable proofs from really proof $\bar{\pi}$. For $\bar{s}_A$, note that the Schnorr proof is just a concatenation of two Schnorr-OR proofs. Therefore, we briefly describe the simulator for Schnor-OR proof. Given $A, \delta \leftarrow \text{dlog}_h B$, Simulator $\mathbb{S}_{OR}$ first samples $c_{A,\mathbb{S}}, z_{A,\mathbb{S}}, r_{B,\mathbb{S}} \overset{\$}{\leftarrow} \mathbb{Z}_q$, sets $a_{B,\mathbb{S}} \leftarrow h^{r_{B,\mathbb{S}}}, a_{A,\mathbb{S}} \leftarrow h^{z_{A,\mathbb{S}}}/A^{c_{A,\mathbb{S}}}$. Now $\mathbb{S}_{OR}$ derives $c_\mathbb{S} \leftarrow H(a_{A,\mathbb{S}}||a_{B,\mathbb{S}}||A||B, h)$, so $c_{B,\mathbb{S}} \leftarrow c_\mathbb{S} \oplus c_{A,\mathbb{S}}$, finally generates $z_{B,\mathbb{S}} \leftarrow r_{B,\mathbb{S}} + c_{B,\mathbb{S}}\delta$. It's trivial to tell that simulated proof shall pass the verification. Note that the special power the $\mathbb{S}_{OR}$ gets is knowledge of $\text{dlog}_h B$ which is unknown to Alice.

We proceed with showing that the simulated proof $\bar{\pi}_\mathbb{S}$ is indistinguishable from the MADV proof $\bar{\pi}$, $\bar{\pi}_\mathbb{S} \approx_c \bar{\pi}$.

Hybrid 0. : starting with $\bar{\pi}_\mathbb{S} = \big((a_{A,\mathbb{S}}, c_{A,\mathbb{S}}, z_{A,\mathbb{S}}), (a_{B,\mathbb{S}}, c_{B,\mathbb{S}}, z_{B,\mathbb{S}})\big)$.

Hybrid 1. : consider a proof $\bar{\pi}^1_\mathbb{S} = \big((a_A, c_A, z_A), (a_{B,\mathbb{S}}, c_{B,\mathbb{S}}, z_{B,\mathbb{S}})\big)$. It is easy to see that the simulated proof $\bar{\pi}_{A,\mathbb{S}}$ is distributed identically to the real $\bar{\pi}_A$. On the one hand $a_{A,\mathbb{S}}$ depends on $z_{A,\mathbb{S}}$ and $c_{A,\mathbb{S}}$ where both $a_{A,\mathbb{S}}$ and $c_{A,\mathbb{S}}$ are uniform in $\mathbb{Z}_q$. On the other hand, since γ is uniform so does a_A because $a_A \leftarrow h^\gamma$. So any *PPT* $\mathbb{A}$ will be unable to distinguish the uniform $a_{A,\mathbb{S}}$ from the uniform a_A. Also, as $c_{A,\mathbb{S}}$ is uniform in $\mathbb{Z}_q$, c is indistinguishable from a random value under a random oracle model, $c_{A,\mathbb{S}}$ wil lbe indistinguishable from c_A which is masked by the random c. Finally, because z_A depends on r, γ, c, where r, c are uniform, it will hold that z_A is uniform as well. Therefore, given $z_{A,\mathbb{S}}$ is sampled from $\mathbb{Z}_q$, $\mathbb{A}$ is incapable of distinguishing two uniform values $z_A, z_{A,\mathbb{S}}$ from each other. In conclusion, we have $\bar{\pi}^1_\mathbb{S} \approx_c \bar{\pi}_\mathbb{S}$.

Hybrid 2. : consider a proof $\bar{\pi}^2_\mathbb{S} = \big((a_A, c_A, z_A), (a_B, c_{B,}, z_B)\big)$. With $a_{B,\mathbb{S}}$ depends on $r_{B,\mathbb{S}}$, where $r_{B,\mathbb{S}}$ is sampled from $\mathbb{Z}_q$, so $a_{B,\mathbb{S}}$ is uniform. Applying the same argument to a_B, we know that the uniform a_B is indistinguishable from $a_{B,\mathbb{S}}$. $c, c_\mathbb{S}$ is uniform because they are outputs of the random oracle. Therefore, c_B, which is masked by c, is indistinguishable from $c_{B,\mathbb{S}}$, which is masked by $c_\mathbb{S}$. z_B, sampled from $\mathbb{Z}_q$, has identical distribution with $z_{B,\mathbb{S}}$ which masked a uniform value $r_{B,\mathbb{S}}$. Therefore, we finalize hybrid 2 with $\bar{\pi}^2_\mathbb{S} \approx_c \bar{\pi}^1_\mathbb{S}$.

With hybrid proof $\bar{\pi}_{\mathbb{S}^1}$ and $\bar{\pi}^2_\mathbb{S}$, we conclude that $\bar{\pi}_\mathbb{S} \approx_C \bar{\pi}$ because $\bar{\pi}_\mathbb{S} \approx_c \bar{\pi}^1_\mathbb{S} \approx_c \bar{\pi}^2_\mathbb{S} = \bar{\pi}$. □

B FUNCTIONALITIES

To prove our NOTRY in the UC model is to construct a simulator $\mathbb{S}$ that emulates all the possible transcript a *Real World* adversary $\mathbb{A}$ can produce. Note that $\mathbb{S}$ executes $\mathbb{A}$ internally. Therefore, any message exchanged between $\mathbb{A}$ and $\mathbf{Z}$ is through a communication channel by $\mathbb{S}$. Also, messages $\mathbb{S}$ received from $\mathbf{Z}$ are copied to $\mathbb{A}$. For a party $\mathbb{P}$ in the *Ideal World*, $\mathbb{S}$ simulates a corresponding party $\mathbb{P}^\mathbb{S}$ in the *Ideal World* for $\mathbb{A}$. $\mathbb{A}$ is a fully adaptive adversary with arbitrary behaviors. $\mathbb{S}$ will compromise $\mathbb{P}$ accordingly after $\mathbb{A}$ corrupts a simulated party $\mathbb{P}^\mathbb{S}$.

Algorithm 1 shows the Ideal functionality constructed for NOTRY-Kex-Ratchet. Parties join the NOTRY-Kex-Ratchet by $\mathbf{Z}$ sending them *init* or *responde* message. A party with *init* message is called initiator I and the one *repondes* is the responder R. Before joining

NOTRY-Kex-Ratchet protocol, all parties are expected to get themselves registered to the $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$ with their identity key pairs, pk, and sk. To simplify simulator construction, F_{kex} guarantees that the *responder* message from R will not be handled until $\mathbb{S}$ sees a *init* message. $\mathbb{S}$ controls delivery of the final shared secret key via an *ok* request. $\mathbb{S}$ later individually delivers the key to parties in the *Ideal World*. We model post-specified peers by allowing parties joining NOTRY without declaring their intended partners and $\mathbb{S}$ specifying the remote identity when delivering the shared secret. $\mathbb{S}$ can arbitrarily deliver the secret if the corresponding party is corrupted.

We model constructiveness by decoupling a *non-information* part from $\mathbb{S}$. The initial analysis on universally composable key exchange protocol [22] observes that the ideal functionality F_{kex} that simply outputs a random key cannot be realized in DH family protocols. This is because an adaptive adversary can corrupt one of two participants before the final message flow to get its ephemeral states. Based on that $\mathbf{Z}$ can distinguish the protocol in the *Ideal World* from the *Real World*. $\mathsf{NonInfo}_{\mathsf{NOTRY}}$, depicted in Algorithm 5, is defined to be part of the ideal functionality F_{kex} to capture the notion of AKE under the Universal Composability Model.

The idea of $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ is to provide $\mathbb{S}$ with a secret internal state to simulate a real protocol. The idea of constructing this functionality is to generate both secret components and then send their corresponding public components to the $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ caller $\mathcal{M}$. $\mathcal{M}$ is the F_{kex} when it is asked to hand a key to a pair of parties. $\mathcal{M}$ can accept or reject(by sending *ok* messege through F_{kex}) the proposed keys. If $\mathcal{M}$ agrees, $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ completes the key exchange process and locally outputs the final shared key. $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ discards them when $\mathcal{M}$ rejects and acts as a responder to finish the key exchange by accepting one public component proposed by $\mathcal{M}$. It's necessary to achieve *initiator resilience* for $\mathbb{S}$ still to be capable of finalizing the protocol on behalf of the remaining honest party. In addition, $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ also maintains an interface for $\mathbb{S}$ to generate SoK messages since internal secret data are all coming from $\mathsf{IncProc}_{\mathsf{NOTRY\text{-}Kex}}$.

According to the composability property, we divided our security proof of NOTRY into two parts: proof of NOTRY-Kex-Ratchet and proof of NOTRY-Avow. We argue that by proving NOTRY-Kex-Ratchet and NOTRY-Avow are UC-secure respectively we can conclude that NOTRY is UC-secure.

C PROOF OF NOTRY-KEX-RATCHET

C.1 Simulator construction

We denote three messages in a round NOTRY-Kex-Ratchet as: m_1^{kex}, m_B^{kex}, m_A^{kex}. Other than the initial round, only m_B or m_A is needed. If any messages sent from $\mathbb{A}$ in the context of the simulated environment are unrelated to NOTRY-Kex-Ratchet, they will be discarded by $\mathbb{S}$. If messages are delayed delivery by $\mathbb{A}$, then $\mathbb{S}$ just waits until messages are delivered. NOTRY-Kex-Ratchet is secure against fully adaptive corruptions. $\mathbb{A}$ is allowed to corrupt any party arbitrarily.

$\mathbb{S}$ begins simulation by initializing the protocol via a message $(setup, \mathbb{G}, q, g, h)$ to $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ through F_{kex}. $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ generates messages for $\mathbb{S}$ to simulate and sends back $(exchange, A, B)$ to $\mathbb{S}$. $\mathbb{S}$ stores the ephemeral values A, B for future use.

$\mathbb{S}$ holds until being activated by an $(init, sid, I, \Phi)$ from F_{kex}. After that, $\mathbb{S}$ compiles a m_1^{kex} for $I^{\mathbb{S}}$ with the material A received from $\mathsf{NonInfo}_{\mathsf{NOTRY}}$. $\mathbb{S}$ broadcasts m_1^{kex} through $\mathbb{A}$ as if it was a messages sent by $I^{\mathbb{S}}$.

After getting a $(responded, sid, rid, R, \Phi)$ from F_{kex}, $\mathbb{S}$ first checks the transmission status of m_1^{kex}. Since F_{kex} only sends a *responded* message after it has already gotten a *init* message, m_1^{kex} must have been sent and delivered in the simulated environment. $\mathbb{S}$ aborts the protocol once m_1^{kex} is not in its format or invalid and marks R as *aborted* at once. If m_1^{kex} is valid then $\mathbb{S}$ constructs the corresponding message m_B^{kex} for $\bar{R}$ in response to m_1^{kex}:

- If m_1^{kex} was created by $\mathbb{S}$ before or was from a corrupted party $I^{\mathbb{S}}$, then based on the messages got from $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ $\mathbb{S}$ constructs the message m_B^{kex}. $\mathbb{S}$ sends a proof request with the identifier of $\bar{R}$ to $\mathsf{NonInfo}_{\mathsf{NOTRY}}$. It's $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ that generates the corresponding signature of knowledge δ_B by γ, x, then $\mathbb{S}$ setup the message m_B^{kex} as (B, δ_B). Note that in the case of corruption, $\mathbb{S}$ will send a (*complete*, *false*,*A*) to $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ to indicate that its transcript has been rejected. F_{kex} immediately sends internal states of $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ to $\mathbb{S}$. $\mathbb{S}$ sends $(ok, sid, rid, 0)$ to F_{kex}, causing it record the output from $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ as the shared key.
- If the received message A' is not identical to the message m_1^{kex} created by $\mathbb{S}$ before, $A' \neq A$, trivially we can see that m_1^{kex} must have been altered by $\mathbb{A}$ during transmission. Therefore, m_B^{kex} will be constructed from $\mathsf{IncProc}_{\mathsf{NOTRY\text{-}Kex}}$. $\mathbb{S}$ sends $(abort, rid, sid)$ to F_{kex} while withhold delivery of the *abort* message to both $\bar{I}$ and $\bar{R}$. Next, $\mathbb{S}$ sends $(incriminate, sid, rid, sid)$ to F_{kex} to invoke $\mathsf{IncProc}_{\mathsf{NOTRY\text{-}Kex}}$. Based on m_1^{kex}, $\mathbb{S}$ sends $(inc, sid, \bar{R}, A')$ to the $\mathsf{IncProc}_{\mathsf{NOTRY\text{-}Kex}}$
process and waits response (inc, sid, B, δ). According to the response from $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ $\mathbb{S}$ complies m_B^{kex} as (B, δ).

$\mathbb{S}$ sends m_B^{kex} through $\mathbb{A}$ as if $R^{\mathbb{S}}$ responses to $I^{\mathbb{S}}$ and sends $(deliver, sid, rid, I, P)$ to F_{kex} to indicate that I generated the shared key.

After receiving message m_B^{kex} from $\mathbb{P}$, $\mathbb{S}$ first checks that if $I^{\mathbb{S}}$ has previously broadcast the message m_1^{kex}. The coming message m_B^{kex} will be ignored if no matching m_1^{kex} is found. If a corresponding record of m_1^{kex} found, $\mathbb{S}$ parses m_B^{kex} to get (B, σ_B). $\mathbb{S}$ will abort the protocol by sending $(abort, sid)$ to F_{kex} if m_B^{kex} is not in an expected form, or m_B^{kex} is not matching the previous message m_1^{kex}. Meanwhile, $\mathbb{S}$ delivers the *abort* message to I and withholds *abort* message to R.

Otherwise, $\mathbb{S}$ constructs m_A^{kex}, which is σ_A, to emulate the message from $\bar{I}$ to $\bar{R}$. In addition, $\mathbb{S}$ will set up the shared secret key. Depending on the situation of protocol emulation, $\mathbb{S}$ constructs m_A^{kex} and outputs a shared key in the following way:

- If $\mathbb{A}$ has already corrupted $\bar{R}$, then the corresponding party $R^{\mathbb{S}}$ must have been corrupted by $\mathbb{S}$ as well. $\mathbb{S}$ simply gets sk_R from $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$ with a retrivesecret request. Next, σ_A is returned from $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ by requesting it with $(prove, I)$. Finally, I is supposed to output a session key with respect

Algorithm 1 Ideal Functionality of NOTRY F_{kex}

F_{kex} proceeds with security parameter λ in the $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$-hybrid model with parties $\mathbb{P}$ and $\mathbb{P}$' and an ideal adversary $\mathbb{S}$. When initializing, F_{kex} calls $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ $\mathcal{N}$ with fresh randomness.

```
1:  On interaction with $\mathcal{N}$:
2:  Forward messages from $\mathbb{S}$ to $\mathcal{N}$. If one of the two is corrupted
    or $R$ is "aborted" while $\mathcal{N}$ has generated its output, reveal
    the state of $\mathcal{N}$ to $\mathbb{S}$. Session id is denoted as sid, round id is
    denoted as rid, $\phi$ is session state.
3:  On (init, sid, I, aux, $\phi$) from $\mathbb{P}$:
4:  if sid is incriminated return
5:  if ($I \neq \mathbb{P}$ ) return
6:  if sid is in null
7:      mark $I$ as the initiator, "active"
8:      mark sid as initialized
9:  record(sid, I) as I
10: Send(init, sid, I, $\phi$) to $\mathbb{S}$
11: On (responde, sid, R, $\phi$) from $\mathbb{P}$:
12: if (sid is not in initialized) return
13: if ($R \neq \mathbb{P}$ ) return
14: if (the session is null) hold until being initialized
15: if sid is initialized
16:     mark $R$ as responder
17:     mark sid as responded
18: init rid with a random value and mark sid as NULL
19: record(sid, rid, R) as II
20: Send(responded, sid, rid, R, $\phi$) to $\mathbb{S}$
21: On (ok, sid, rid, k) from $\mathbb{S}$:
22: if (the record III of (rid, sid, K) found) return
23: //Test the key of this round is ready or not, only allow to set
    //it once.
24: if (($I \wedge R$) are corrupted) return
25: //If both are corrupted, simulation is handled by the $\mathbb{A}$
26: if ($I \wedge R$ are uncorrupted)
27:     $K \xleftarrow{\$} \{0,1\}^\lambda$
28: //If both are honest, they output a random key, this captures
    //both PCS and forward secrecy
29: else if ($R$ is corrupted)
30:     {$K$ be the local output of $\mathcal{N}$}
31: //This captures initiator resilience, whereas the output key
    //is random or the local output of $\mathcal{N}$ if $R$ is honest.
32: else {$K \leftarrow k$}
33: record(sid, rid, K) as III
34: On (deliver, sid, rid, $\mathbb{P}$, $\mathbb{P}'$) from $\mathbb{S}$:
35: if (sid is not "responded" ∨ no record III related to ($\mathbb{P}$, $\mathbb{P}$' )
    return
36: if (a set-key message sent to $\mathbb{P}$) return
37: if (rid is "incriminated" ∨ "exchanged") return
38: if ($\mathbb{P} \notin$ (initiator, responder) ) return
39: if (($\mathbb{P}$, $\mathbb{P}'$) $\neq$ (initiator, responder) ∧ ($\mathbb{P}$ is not corrupted))
    return
40: Send(set-key, sid, rid, $\mathbb{P}$, K) to $\mathbb{P}'$
41: //This captures post-specified peer since the id of the peer is
    //part of the output
42: if (two set-key have been sent)
43:     mark rid as "exchanged"
44: Switch roles $(I, R) \longleftarrow (\mathbb{P}', \mathbb{P})$
45: Join records I, II, III as (sid, rid, I, R, K)
46: Inc round number $rid \longleftarrow rid + 1$
47: //For multi-round key exchange
48: On (abort, rid, sid) from $\mathbb{S}$
49:     Send delayed (abort, sid, I) to I
50:     Send delayed (abort, sid, R) to R
51: On (incriminate, sid, rid) from $\mathbb{S}$
52: if (rid is "incriminated" ∨ "exchanged") return
53: if ((R is "aborted") ∧ (I is "active") ∧ (R is uncorrupted))
54:     Send(retrieve-secret, R) to $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$
55:     $sk_R \leftarrow \mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$.retrieve-secret(R)
56:     Execute IncProc(sid, I, R, $pk_I$, $pk_R$, $sk_R$)
57:     mark sid as incriminated
```

to the one distributed between $I^{\mathbb{S}}$ and $P^{\mathbb{S}}$. If $\mathbb{S}$ generated a message m_B^{kex} for party $R^{\mathbb{S}}$ but $R^{\mathbb{S}} \neq P^{\mathbb{S}}$, $\mathbb{S}$ aborts the F_{kex} and withholds the *abort* message to parties. Otherwise, F_{kex} exposes the internal state of $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ to $\mathbb{S}$ due to corruption, which is the secret component γ for generating A. Based on γ, $\mathbb{S}$ computes the $K = B^\gamma$. Combined with K, generated σ_A, and received σ_B $\mathbb{S}$ derives the final session key k_{sess} and sets up protocol outputs as I, P, k_{sess} by sending (ok, sid, K) to F_{kex}.

- Otherwise, SoK σ_B is valid iff it's generated with the knowledge of the corresponding secret key sk_R or discrete log of AB to the base g. Furthermore, σ_B is fully bounded to this instance of NOTRY-Kex-Ratchet because it depends on the exchanged message A, B. In this case, we can conclude that $\mathbb{S}$ generated the message m_B^{kex} and $P = R^{\mathbb{S}}$. So $\mathbb{S}$ get the simulated σ_A based on the requesting $\mathsf{NonInfo}_{\mathsf{NOTRY}}$ with $(prove, sid, I)$. After that, $\mathbb{S}$ forces the protocol to output a random shared key by sending $(ok, sid, 0)$ to F_{kex}.

$\mathbb{S}$ sends m_A^{kex} through $\mathbb{A}$ as if $I^{\mathbb{S}}$ sent it to $I^{\mathbb{S}}$.

When $R^{\mathbb{S}}$ gets back the SoK σ_A of $\bar{I}$, $\mathbb{S}$ examines that if $\bar{R}$ has already saw message m_1^{kex} from $I^{\mathbb{S}}$ and it has sent a corresponding response m_B^{kex}. m_A^{kex} will be ignored if either of the above requirements failed. $\mathbb{S}$ validates σ_A and then outputs the distributed key.

- if σ_A fails verification, which implies the σ_A might be altered by $\mathbb{A}$ or it does not match previous messages. In this case, $\mathbb{S}$ sends $(abort, sid)$ to F_{kex} and delivers *abort* message to R at once while withholding the abort message to I.

Algorithm 2 Ideal Functionality of NOTRY-Avow F_{avow}
F_{avow} is parameterized with ADV proof π, designated verifier *Judge*, public component(s) to be avowed [AB], and the two parties corresponding secrets for the ADV proof. *IdealAvow* signals $\mathbb{S}$ the validation of avowal by sending *fail* or *proof* message.

```
1:  On (Setup,sid, $\mathbb{G}$, $q$, $h$, $g$, $\ell$) from $\mathbb{S}$:
2:    if sid is already setup return
3:    else Setup the group $\mathbb{G}$ with respect to its generators $g, h, \ell$
4:    mark sid as setup
5:  On (Avow, sid, $pk_j$, [secret], [AB]) from $\mathbb{P}$:
6:    if (matched record of [AB] and sid found $\wedge$ $\mathbb{P} \neq \mathbb{P}'$)
7:    //Mutually-agreed avowal is captured by holding the avowal
      //until $\mathbb{P}$ and $\mathbb{P}$' both request to avow.
8:      if ($h^{[secret_{\mathbb{P}}]+[secret_{\mathbb{P}'}]} \neq [AB]$)
9:        Send(fail, [AB], $pk_j$, $\mathbb{P}$, $\mathbb{P}'$) to $\mathbb{S}$
10:     //Decides if it's the witness to $\mathrm{dlog}_h AB$
11:     else
12:       $sk_j \leftarrow \mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$.retrieve-secret($j$)
13:       Send(proof, AB, $sk_j$, Output, $\mathbb{P}$, $\mathbb{P}'$) to $\mathbb{S}$
14:   //Indicating $\mathbb{S}$ the result of an (in)successful avow from $\mathbb{P}$ and
      $\mathbb{P}'$ and deliver the $sk_j$ to $\mathbb{S}$ for simulation.
15:   else record(sid, $pk_j$, [$secret_{\mathbb{P}'}$], [AB], $\mathbb{P}' \leftarrow \mathbb{P}$) and hold
16:   //To capture mutually-agreed avow, protocol proceeds only
      //after accepting two avow requests from two different parties
      //with the same AB's. The party that starts avowal is recorded
      //as $\mathbb{P}'$.
17: On (Judge, $\pi$, [AB], $pk_j$) from $\mathbb{P}$:
18:   $b \leftarrow$NOTRY-Judge ($\pi$, [AB], $pk_j$) // verify the proof
19:   Send($\pi$, [AB], $pk_j$, $b$) to $\mathbb{P}$.
```

Algorithm 3 Incriminate Procedure $\mathsf{IncProc}_{\text{NOTRY-Kex}}$
Subroutine $\mathsf{IncProc}_{\text{NOTRY-Kex}}(sid,\ I,\ R,\ sk_R)$

```
1: On (inc, sid, I, R, $\mathbb{G}$, q, $g$, $h$, A ) from $\mathbb{S}$:
2:   Generate $\delta \xleftarrow{\$} \mathbb{Z}_q$, $B \leftarrow h^{\delta}$
3:   $\sigma_R \leftarrow$ SoK-Sign( $(A, B), \bar{S}_B, \delta, sk_R$)
4:   Send(inc, sid, B, $\sigma$) to $\mathbb{S}$
5:   //Working as B as contributing his share to the exchanged
     key
```

- if σ_A is valid, $\mathbb{S}$ outputs a key for R as well. A secure valid SoK indicates that m_A^{kex} is sent from a corrupted I or protocol executed honestly. In both cases, $\mathbb{S}$ has already instructed F_{kex} to record a session key. $\mathbb{S}$ completes by sending $(deliver, sid, rid, R, I)$ to F_{kex} to sends indicate that R generated the shared key.

When $\mathbb{A}$ instructs $I^{\mathbb{S}}$ to broadcast m_1^{kex}, but $\mathbb{S}$ has not gotten a *init* message from F_{kex} yet, then $\mathbb{S}$ awaits I to be activated by Z with $(init, sid, I, \perp, \perp)$ to F_{kex} for simulation but discards any *init* message from F_{kex}. When $\mathbb{A}$ instructs $R^{\mathbb{S}}$ to issue message m_B^{kex} while $\mathbb{S}$ has not get a *responded* message yet, then $\mathbb{S}$ awaits R to be activated by Z with $(responde, sid, \perp, R, \perp)$ and discards any *responded* from F_{kex}.

Algorithm 4 Share Functionality $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$
Subroutine $\mathcal{G}_{krkro}^{\varphi,n,\mathbb{G},q,h,g}$. Parameterized by an implicit security parameter λ, a group $\mathbb{G}$ generated by g with prime order q, and a number of observable and programmable random oracles n

```
1:  On (register, sk) from $\mathbb{P}$ :
2:    if ($\mathbb{P}$ is corrupted $\vee$ $\mathbb{P}$ is registered) return
3:    if (sk is null) $sk \xleftarrow{\$} \mathbb{Z}_q$
4:    //Registration with or without sk
5:    $pk \leftarrow g^{sk}$
6:    record("registered", $\mathbb{P}$, pk, sk)
7:    Send(pk) to $\mathbb{P}$
8:  On (retrieve, $\mathbb{P}'$) from $\mathbb{P}$:
9:    if (record tuple of $\mathbb{P}'$ found)
10:   //Registered public keys are available to anyone.
11:     Send(pub-key, $\mathbb{P}'$, $pk_{\mathbb{P}'}$) to $\mathbb{P}$
12:   else Send(pub-key, $\mathbb{P}'$, $\perp$) to $\mathbb{P}$
13: On (retrieve-secret, $\mathbb{P}'$) from $\mathbb{P}$
14:   if ($\mathbb{P} \neq \mathbb{P}'$ ) return
15:   //Only allows the owner to get the corresponding secret key.
16:   if ($\mathbb{P}$ is registered) Send(sk-key, $\mathbb{P}'$, $pk_{\mathbb{P}'}$, $sk_{\mathbb{P}'}$) to $\mathbb{P}$
17:   else Send(sk-key, $\mathbb{P}'$, $\perp$, $\perp$) to $\mathbb{P}$
18: On (random-oracle, i, x, sid ) from $\mathbb{P}$:
19:   if ($i \notin [1, n]$) return
20:   //Reject an out of scope random oracle request .
21:   if (record of x found) //x is requested before
22:     Send(d) to $\mathbb{P}$
23:   else $d \xleftarrow{\$} \{0,1\}^{\lambda}$
24:   record(i, x, d, $\mathbb{P}$)
25:     Send(d) to $\mathbb{P}$
26: On (observe-RO, i, $\mathbb{P}$) from $\mathbb{S}$:
27:   Send(x, d) of the corresponding $\mathbb{P}$ back to $\mathbb{S}$
28:   //$\mathbb{S}$ is allowed to inspect the request record of any $\mathbb{P}$.
```

C.2 Indistinguishable realization

We prove that F_{kex} EUC-securely realized NOTRY-Kex-Ratchet by proving that $\mathbb{S}$ interacting with F_{kex} is indistinguishable from $\mathbb{A}$ interacting with NOTRY-Kex-Ratchet. We will show that for any attacks $\mathbb{A}$ can do in the *Real World* there is a simulator $\mathbb{S}$ emulating them in the *Ideal World* such that:

Lemma 6. $IDEAL_{\mathsf{F}_{kex}, \mathbb{S}, \mathsf{Z}} \overset{c}{\approx} REAL_{\text{NOTRY-Kex-Ratchet}, \mathbb{A}, \mathsf{Z}}$

To prove Lemma 6 is to show that the simulator $\mathbb{S}$ we described above computationally emulates an indistinguishable ideal execution view for all the possible execution views in the *Real World*. Our proof proceeds with demonstrating emulation under the honest case and the corruption case.

C.2.1 The honest case. This happens when $\mathbb{A}$ does not alter any message or corrupt any party until the session ends. Therefore, three message flows were generated by $\mathbb{S}$ with the help of $\mathcal{N}$. Note that in this case the two SoKs were not generated by proving knowledge of any secret keys. They were produced by the DL of AB to the base g. We argue that Z is unable to distinguish a simulated SoK

Algorithm 5 Non-Information Oracle NonInfo$_{\text{NOTRY}}$ $\mathcal{N}$
Subroutine NonInfo$_{\text{NOTRY}}$

```
1:  On (setup, $\mathbb{G}$, $q$, $h$, $g$, $l$ ) from $\mathcal{M}$:
2:    if (the first setup message)
3:      parse $\mathbb{G}$, $q$, $h$, $g$, $l$ as group $\mathbb{G}$, prime order $q$, and generator $h$, $g$, $l$
4:    else discard parameters
5:    generate $(\gamma,\kappa) \xleftarrow{\$} \mathbb{Z}_q$, $(A, AB, B) = (h^\gamma, g^\kappa, AB/A)$
6:    record("setup", $\mathbb{G}$, $q$, $h$, $g$, $l$, $\gamma$)
7:    Send(exchange, $A, B, \mathbb{G}, q, h, g, l$) to $\mathcal{M}$
8:  On (complete, ok, $A, \sigma_A, \sigma_B$) from $\mathcal{M}$:
9:    if (not setup yet) return
10:   if (ok is True $\wedge\ \sigma_A \wedge\ \sigma_B$)
11:     set $K$ = KDF $(A \,||\, B \,||\, \sigma_A \,||\, \sigma_B \,||\, B^\gamma)$
12:   else if $((A \notin \mathbb{G}) \vee (A$ is an identity element$))$ return
13:   else
14:     generate new $\kappa \xleftarrow{\$} \mathbb{Z}_q$, $AB \leftarrow g^\kappa$, $B \leftarrow AB/A$
15:     generate $\sigma_A$ and $\sigma_B$ with $(\gamma, \kappa)$
16:     set $K$ = KDF$(A \,||\, B \,||\, \sigma_A \,||\, \sigma_B \,||\, g^{\gamma\kappa}/A^\gamma)$
17:   locally output $K$
18: On (prove, $\mathbb{P}$, $sk_{\mathbb{P}}$) from $\mathcal{M}$:
19:   if (not setup) return
20:   if ($\mathbb{P}$ $\notin$(initiator,responder)) return
21:   if ($sk_{IR}$) $\sigma \leftarrow SoK(A, B, pk_{\mathbb{P}}, AB, \gamma, sk_{\mathbb{P}})$
22:   else generate $\sigma \leftarrow SoK(A, B, pk_{\mathbb{P}}, AB, \gamma, \kappa)$
23:   Send(proof, $\sigma$) to $\mathcal{M}$
```

from $\mathbb{S}$ between a real SoK in actual interactions. **Z** learns nothing beyond the fact the two SoKs are both valid for the same statement.

The full transcript simulated by $\mathbb{S}$ is $(A^{\mathbb{S}}, B^{\mathbb{S}} \leftarrow (AB^{\mathbb{S}})/A^{\mathbb{S}}, \sigma_A^{\mathbb{S}}, \sigma_B^{\mathbb{S}})$ and the real transcript is $(A, B, \sigma_A, \sigma_B)$. Note that $I^{\mathbb{S}}$ will abort after m_B^{kex} is sent if **Z** tries to provide an inconsistent session state$(\Phi_I \neq \Phi_R)$. The output of the protocol will be identical to the real one because σ_B will is invalid to $\bar{I}$ in the *Real World*.

C.2.2 Dishonest cases.

- Modified m_1^{kex}: this happens when m_1^{kex} from $\mathbb{S}$ is modified by $\mathbb{A}$ in transmission, but neither $I^{\mathbb{S}}$ nor $R^{\mathbb{S}}$ is corrupted until m_1^{kex} is delivered. In this case, $\mathbb{S}$ generates $\sigma_B^{\mathbb{S}}$ by IncProc$_{\text{NOTRY-Kex}}$. IncProc$_{\text{NOTRY-Kex}}$ generates $\sigma_B^{\mathbb{S}}$ as the way $\bar{R}$ generates it in the *Real World*.
- Modified m_B^{kex}: this happens when $\mathbb{A}$ modifies m_B^{kex} generated by $\mathbb{S}$ but both of them are not corrupted when m_B^{kex} is transmitted. $I^{\mathbb{S}}$ will immediately know the modification after it checks $\sigma_B \in m_B^{kex}$ since the verification will fail. This is also identical to the case in *Real World* of $\bar{I}$.
- Modified m_A^{kex}: this happens when $\mathbb{A}$ modifies m_A^{kex} from $\mathbb{S}$ while neither $I^{\mathbb{S}}$ nor $R^{\mathbb{S}}$ are corrupted when m_B^{kex} is delivered. $\mathbb{S}$ instructs $R^{\mathbb{S}}$ to abort and $\bar{R}$ in the *Real World* will abort too since σ_A is unable to pass the verification.

So the transcript output by $\mathbb{S}$ in this case will be $(A_1^{\mathbb{S}}, B_1^{\mathbb{S}}, \sigma_{B1}{}^{\mathbb{S}}, \sigma_{A1}{}^{\mathbb{S}})$ where only $\sigma_{A1}{}^{\mathbb{S}}$ is generated through proof simulation.

C.2.3 *Corruption.*

This is the case when one of the parties is corrupted during times that are not covered in dishonest cases. In terms of transcript, the advantage of corruption is access to the secret key. Therefore, the corresponding SoK, say σ_A for $I^{\mathbb{S}}$, is generated by taking the secret key as one of the two witnesses in the SoK statement.

When both parties are corrupted, $\mathbb{S}$ does nothing but copy any outputs from parties in the *Ideal World* to parties in the *Real World*. So there is no simulation at all, for **Z** to distinguish.

The simulated transcript of $\mathbb{S}$, in this case, is $(A_2^{\mathbb{S}}, B_2^{\mathbb{S}}, \sigma_{A2}{}^{\mathbb{S}}, \sigma_{B2}{}^{\mathbb{S}})$ where one of two SoKs is generated with the secret key of the corrupted party.

C.2.4 *Hybrid Arguments.*

We start hybrid arguments with the transcript in the honest case.

Proof. *(Hybrid 0)* is the transcript $\mathcal{T}_0^{kex} = (A^{\mathbb{S}}, B^{\mathbb{S}} \leftarrow (AB^{\mathbb{S}})/A^{\mathbb{S}}, \sigma_A^{\mathbb{S}}, \sigma_B^{\mathbb{S}})$. where both SoKs are simulated.

Hybrid 1: replace $\sigma_B^{\mathbb{S}}$ in $\mathcal{T}_0^{kex}$ with $\sigma_{B1}{}^{\mathbb{S}}$ to get $\mathcal{T}_1^{kex}$. $\mathcal{T}_0^{kex}$ is indistinguishable from $\mathcal{T}_1^{kex}$ because **Z** is unable to distinguish a simulated valid SoK(without secret key) from a real SoK(with secret key). This is what $\mathbb{S}$ simulated in the Dishonest cases

Hybrid 2: rather than replace $\sigma_B^{\mathbb{S}}$ in $\mathcal{T}_0^{kex}$, replace $\sigma_A^{\mathbb{S}}$ with $\sigma_{A1}{}^{\mathbb{S}}$, generated with knowledge of the secret key, to get $\mathcal{T}_2^{kex}$. The same argument in $\mathcal{T}_1^{kex}$ can be applied to $\mathcal{T}_2^{kex}$ and $\mathcal{T}_0^{kex}$. It's obvious to see that $\mathcal{T}_1^{kex}$ and $\mathcal{T}_2^{kex}$ capture the output of $\mathbb{S}$ in the corruption case. That is, one of two SoKs is not simulated.

Hybrid 3: replace $\sigma_A^{\mathbb{S}}$ in $\mathcal{T}_1^{kex}$ with $\sigma_{A2}{}^{\mathbb{S}}$, which is generated with the knowledge of the secret key, to get $\mathcal{T}_3^{kex}$. Note that the only distinction between $\mathcal{T}_3^{kex}$ and $\mathcal{T}_1^{kex}$ is that $\sigma_{A2}{}^{\mathbb{S}}$ in $\mathcal{T}_3^{kex}$ is proven by knowing the corresponding secret key whereas $\sigma_A^{\mathbb{S}}$ in $\mathcal{T}_1^{kex}$ is simulated. Again, **Z** is unable to tell a simulated SoK from a real SoK to the same statement. So $\mathcal{T}_3^{kex}$ is indistinguishable from $\mathcal{T}_1^{kex}$. Observe that this is exactly the case in the *Real World* execution.

All "A"s and "B"s in all the transcripts($\mathcal{T}_{0-3}^{kex}$) are just DH tuples, since we assume DDH holds, all the "A"s and "B"s in all the transcripts should be indistinguishable from each other. That, we conclude that **Z** is unable to distinguish $\mathbb{S}$ in the *Ideal World* from $\mathbb{A}$ in the *Ideal World*. □

D PROOF OF NOTRY-AVOW

D.1 Simulator construction

Proof. For convenience, we denote the three messages in NOTRY-Avow as $(m_1,\ m_2,\ m_3)$. First, **Z** activates $\mathbb{S}$ to initialize the NOTRY-Avow protocol by sending a *setup* to $\mathcal{N}$ through F$_{kex}$ and waits for an *exchange* response. After receiving an *exchange* message from $\mathcal{N}$, $\mathbb{S}$ and $\mathbb{A}$ finish another round of NOTRY-Kex-Ratchet to acquire a temporary session key for future use in NOTRY-Avow. **Z** instructs parties to start avowal by sending *Avow* message to F$_{avow}$. Note here we assume the identity of *Judge* is common knowledge since we modeled *Judge* as part of the NOTRY-Avow. The arbitrary inputs of parties, [*secret*]s, and public components [*AB*]s are from **Z**.

F_{avow} holds until it gets another *Avow* message. F_{avow} proceeds only after it receives two *Avow* messages with identical $[AB]$s, which captures that both parties mutually request to avow the same part of their conversation. After getting the second *Avow*, F_{avow} matches two records that share $[AB]$ values and tests if the avowal is successful or not. If this is a valid avowal, F_{avow} simulates session key exchanged k_{rel}, simulated transcripts of avowal $m_1(E_A, R_A), m_2 = (R_B, \mathsf{Enc}(k_{\text{rel}}, (c_A, z_A, s_A))), m_3(\mathsf{Enc}(k_{\text{rel}}, (c_B, z_B)))$, and keep the rest of O as the ADV avowal proof π. $\mathbb{S}$ receives all the simulated materials from F_{avow} and stores them for later emulation. If this is a failed avowal, F_{avow} sends *fail* message to $\mathbb{S}$. Note that $\mathbb{S}$ knows distributed session key in the *Ideal World* since it completes the emulation. In that case, $\mathbb{S}$ simulates the critical part of avowal z_A and z_B by arbitrarily sampling the z_A and z_B which will lead to a failed avowal.

Next, $\mathbb{S}$ honestly constructs m_1 in NOTRY-Avow with the materials it gets from F_{avow} when it observes the corresponding message sent from $\bar{I}$ to $\bar{R}$. $\mathbb{S}$ sends (E_A, R_A) to the simulate party in the *Real World*, $R^{\mathbb{S}}$, through $\mathbb{A}$ to emulate this behavior. $\mathbb{S}$ constructs $m_2 = (R_B, \mathsf{Enc}(k_{\text{rel}}, (c_B, z_B)))$ for NOTRY-Avow in response to I based on the message even if it's different from the one it got from $\mathcal{N}$ and sent it to R. $\mathbb{S}$ just forward m_2 from $R^{\mathbb{S}}$ to $I^{\mathbb{S}}$ as the internal real-world $\bar{P}$ replies $\bar{I}$ with real m_2. To simulate m_3 message as $\bar{I}$ sends it to $\bar{R}$ in the *Real World*, $\mathbb{S}$ just passes message m_3 from $R^{\mathbb{S}}$ to $I^{\mathbb{S}}$. When $\mathbb{S}$ sees two parties' final outputs in the *Real World*, $\mathbb{S}$ forward the corresponding output, which is z_α, z_β, to two dummy parties in the *Ideal World*. Note that m_1, m_2, m_3 can all be modified by $\mathbb{A}$.

D.2 Indistinguishable realization

F_{avow} UC-realizes NOTRY-Avow iff $\mathbb{S}$ emulates an indistinguishable execution view in the *Ideal World* for any adversary $\mathbb{A}$ in the *Real World*, such that:

Lemma 7. $IDEAL_{\mathsf{F}_{avow}, \mathbb{S}, \mathbf{Z}} \overset{c}{\approx} REAL_{\text{NOTRY-Avow}, \mathbb{A}, \mathbf{Z}}$

To formally prove Lemma 7, we will see how the constructed $\mathbb{S}$ emulates a computationally indistinguishable view adaptively according to the behavior of $\mathbb{A}$. For an adversary $\mathbb{A}$ that can modify, abort, and corrupt parties. If any of the three messages in the *Real World* was modified by $\mathbb{A}$, $\mathbb{S}$ replaces the corresponding message with a random value. Note that under this circumstance with a secure encryption scheme and the DLP assumption holds, the final judgment from F_{avow} on the proof π will be false in both worlds.

For adaptive adversaries, $\mathbb{S}$ has to make subsequent messages consistent with the previous messages even with arbitrary behaviors of $\mathbb{A}$. When $\mathbb{A}$ corrupts a party $\bar{\mathbb{P}}$ in the *Real World*, $\mathbf{Z}$ and $\mathbb{S}$ will immediately know that the $\bar{\mathbb{P}}$ is corrupted. $\mathbb{S}$ generates internal states of the simulated party $\mathbb{P}^{\mathbb{S}}$. If I is corrupted,$\mathbb{S}$ asks F_{avow} for I's secret, c_A, z_A, s_A. $\mathbb{S}$ will construct m_2 as usual since m_2 is completely independent of the m_1. $\mathbb{S}$ watches the behavior of $\bar{\bar{R}}$ in the *Real World*, if $\bar{\bar{R}}$ aborts, then $\mathbb{S}$ just randomly picks a value at the same length of ciphertext m_3. Otherwise, observing $\bar{\bar{R}}$ outputting his part of the proof z_β, $\mathbb{S}$ sends the message m_3 as it is in the honest case.

If both parties were corrupted, by taking advantage of witnesses of the ADV proof avowal, $\mathbb{S}$ simply run the protocol in the *Real World* and get the execution log instead of interacting with F_{avow}. $\mathbb{S}$ simulates message exchanging of parties by running NOTRY-Avow in the *Real World* internally. Note that $\mathbf{Z}$ is unable to distinguish $\mathbb{S}$ from $\mathbb{A}$ since $\mathbb{S}$ works exactly as $\mathbb{A}$ in NOTRY-Avow.

D.2.1 Hybrid Arguments. For the one-party corruption case, the $\mathbf{Z}$ is unable to distinguishable its communication with $\mathbb{A}$ in the *Real World* and $\mathbb{S}$ in the *Ideal World* because of the zero-knowledge properties of ADV proof of avowal. As we've already proved in the Lemma 4. For the honest case and two-party corruption case, we're gonging to prove the transcripts between the two worlds are indistinguishable. Recall the two transcripts are $\mathcal{T}_{\mathbb{S}} = (E'_A, R'_A, R'_B, \bar{c'}_1, \bar{c'}_2, z'_\alpha, z'_\beta, c'_J, z'_J)$ in the *Ideal World* and $\mathcal{T} = (E_A, R_A, R_B, \bar{c}_1, \bar{c}_2, z_\alpha, z_\beta, c_J, z_J)$ in the *Real World*.

(Hybrid 0) is the genuine transcript $\mathcal{T}_{\mathbb{S}}$.

(Hybrid 1) slightly changes $\mathcal{T}_{\mathbb{S}}$ to $\mathcal{T}_1 = (E'_A, R_0, R_1, \bar{c'}_1, \bar{c'}_2, z'_\alpha, z'_\beta, c'_J, z'_J)$, where $(R_0, R_1) \xleftarrow{\$} \mathbb{Z}_q$, suppose there exists a distinguisher $\mathcal{D}$ with advantage ϵ_{DDH} to break DDH assumption, then the advantage of $\mathcal{D}$ to distinguish $\mathcal{T}_{\mathbb{S}}$ and $\mathcal{T}_1$ is bounded by $2\epsilon_{DDH}$.

(Hybrid 2) $\mathcal{T}_2$ will be a single replacement of E'_A with the Pedersen Commitment E_A in $\mathcal{T}_1$. As Pedersen Commitments are perfect hiding, $\mathcal{D}$ is unable to distinguish E_A between E'_A.

(Hybrid 3) the two ciphertexts $\bar{c}'_1, \bar{c}'_2$ in $\mathcal{T}_2$ can be substituted the $\bar{c}_1, \bar{c}_2$ of $\mathcal{T}$ to constitute $\mathcal{T}_3$. Let's say the advantage of $\mathcal{D}$ to break the notion of semantic security is ϵ_{se}. Therefore, we can conclude that $\mathcal{D}$ distinguishes $\mathcal{T}_2$ and $\mathcal{T}_3$ with $2\epsilon_{se}$.

(Hybrid 4) $\mathcal{T}_4$ transfers c'_j, z'_j in $\mathcal{T}_3$ to the two random values (c_j, z_j) in $\mathcal{T}_3$. Since $(c'_j, z'_j), (c_j, z_j)$ are all dependent on two random numbers contributed by I and R respectively. Naturally, under the one-time-pad, the $\mathcal{D}$ is not capable of distinguishing the pairs from each other. That implies $\mathcal{T}_3$ and $\mathcal{T}_4$ are indistinguishable.

(Hybrid 5) observe that z'_α is distinguishable from $c_{AB}\alpha + r$ because r, r_A are randomly sampled from $\mathbb{Z}_q$. Assuming a random oracle, it's obvious that the output of the random oracle c is indistinguishable from r as well. As c_{AB} is masked by c, according to the one-time-pad, we can conclude that c_{AB} is indistinguishable from a random number. Therefore, z'_α is indistinguishable from $r_1\alpha + r_0$. Same arguments applies to z_α and (z'_β, z_β). With z_α and z'_α both indistinguishable from $r_1\alpha + r_0$, it's trivial to see $z_\alpha(z_\beta)$ is indistinguishable from $z'_\alpha(z'_\beta)$. This directly leads $\mathcal{T}_4 = (E_A, R_0, R_1, c_1, c_2, z'_\alpha, z'_\beta, c_J, z_J)$ to be inherently distinguishable from $\mathcal{T}_5 = (E_A, R_0, R_1, c_1, c_2, z_\alpha, z_\beta, c_J, z_J)$.

(Hybrid 6) $\mathcal{T}_6$ is identical to $\mathcal{T}$. follow the same arguments we used in *Hybrid 1*, we can get that the advantage of $\mathcal{D}$ to distinguish $\mathcal{T}_5$ and $\mathcal{T}$ is less than $2\epsilon_{DDH}$. Finally, for any *PPT* distinguisher $\mathcal{D}$, we have

$$|Pr[\mathcal{D}(\mathcal{T}) = 1] - Pr[\mathcal{D}(\mathcal{T}_{\mathbb{S}})]| \leq 4\epsilon_{DDH} + 2\epsilon_{se} \tag{D.1}$$

with all DDH, semantic security, and random oracle holds, both of $\epsilon_{DDH}, \epsilon_{se}$ are negligible. In conclusion, $\mathcal{D}$ only has negligible advantage to distinguish $\mathcal{T}$ from $\mathcal{T}_{\mathbb{S}}$, which makes $\mathcal{T}$ is computationally indistinguishable from $\mathcal{T}_{\mathbb{S}}$. This completes the proof. □