Legally keeping secrets from mobile operators: Lawful Interception Key Exchange (LIKE)

Ghada Arfaoui[1], Olivier Blazy[3], Xavier Bultel[4],
Pierre-Alain Fouque[2], Adina Nedelcu[1,2], and Cristina Onete[3]

[1] Orange Labs
{ghada.arfaoui, adina.nedelcu}@orange.com
[2] Rennes Univ., IRISA, CNRS UMR 6074
pierre-alain.fouque@irisa.fr
[3] University of Limoges, XLIM, CNRS UMR 7252
{olivier.blazy, maria-cristina.onete}@unilim.fr
[4] LIFO, INSA Centre Val de Loire, Université d'Orléans,
xavier.bultel@insa-cvl.fr

Abstract. How to balance user privacy with a law enforcement agency's need to view their communications during investigations has always been a delicate and hard to formalize problem. In the age of analog communication, interception was as simple as "wiretapping" a telephone line. Due to technological advances, this is now mandated, at least in the context of mobile communications, by laws and standards pertaining to lawful interception. We introduce a new primitive, called Lawful Interception Key-Exchange (or LIKE), that guarantees key-security (up to a collision of $n-1$) authorities, as well as a series of novel properties, such as non-frameability and honest operator. Our approach is generic enough to be adapted to a number of use cases, is more privacy preserving than existing implementations and strikes the right balance between security and exceptional access.

1 Introduction

A fundamental type of service that mobile users require from their operator is the ability to communicate with other users, by calls or messaging. In third and fourth-generation networks, the communication between two users goes via their mobile operators, as described in Figure 1. Notably, when Alice initiates a conversation, she establishes a secure channel with her operator via an authenticated key-exchange protocol called AKA. That operator contacts Bob's operator, over yet another secure channel. Finally, Bob's operator establishes a secure channel with Bob, usually once more by using the AKA protocol.

Subsequently, the two operators forward all the communication between Alice and Bob.

The security of AKA has been analyzed and proven to be imperfect [4]. However, even a perfectly secure and forward-secure protocol would not provide end-to-end security for Alice and Bob, since the operator is privy to all their communications. A solution is for Alice and Bob to first encrypt their communications with a key exchanged out-of-band. However, that might be impractical and would require knowledge that a non-expert user does not possess.

To this day, no mobile network offers end-to-end-secure communications as a service, principally for two reasons. One is billing: the operator must know which type of service is demanded by which user and towards which user in order to know how to bill its customers. Another reason is *lawful interception*.

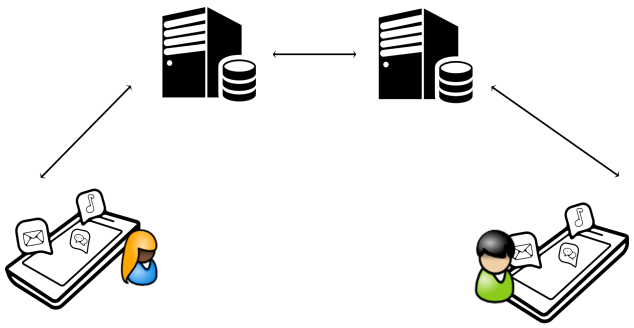

Fig. 1. Alice communicates with Bob via her operator (left) and Bob's operator (right). All the established channels are secure.

Lawful interception (LI) is a legal requirement in many countries and compliance with its regulations is required by technical specifications. A number of legislations such as CALEA [20] and the Council Resolution of 17 January 1995 on the lawful interception of telecommunications [31] stipulate that law enforcement agencies be able to access, under certain conditions, the content of communications and metadata of mobile users. Baker McKenzie presented a comparison of surveillance laws in various countries in [26].

Following suit, the 3rd Generation Partnership Project (3GPP) has developed a series of technical specifications concerning the requirements, the architecture, and the interface of lawful interception (see [1–3] for the most recent documents concerning 5G), which were then adopted by standardization organizations such as ETSI. The architecture of Lawful Interception, detailed in [2], contains three important actors: a 3GPP Communication Service Provider (the operator), a law-enforcement agency (LEA), and the target. The existence of a fourth entity, providing the LEA with a warrant, can be intuited, but its role is less clear-cut. In a nutshell, LI captures the fact that legal authorities (such as a court of justice) may enable law-enforcement agencies (such as the police) to require from the mobile operator the data of any targeted mobile user (say Alice). This data may include secrets shared between the operator and the user, but also associated meta- or user data, including past messages, phone calls, communication partners, etc. Usually, to meet this requirement, the traffic flowing through the operator's network appears either in plaintext or is encrypted with a key known to the operator. Thus, in being obliged to (be able to) infringe the privacy of *specific target* users, the mobile operator currently infringes the privacy of *all* its users.

Our present work is motivated by the belief that privacy is precious and ultimately a fundamental right. This important realization has fueled many privacy-preserving initiatives, including the General Data Protection Regulation (GDPR[5]) and the current international debate on privacy-preserving contact-tracing apps for COVID19. In the context of mobile communications, we believe that while lawful interception – when used correctly and within the boundaries of impartial laws – may be an asset to national security, it should not come at the expense of the privacy of innocent users. Indeed, achieving end-to-end confidentiality even with respect to mobile operators is an important step towards cryptographically resisting mass-surveillance. Our goal is therefore to give operators the tools and abilities to protect their users to the fullest possible degree, within the limits of the law. We restrict ourselves in this paper to the setting in which both our operators are running their services in the same country (and lawful-interception system).

[5] `https://gdpr.eu/tag/gdpr/`

Our contributions In this paper we show a way to provide as much user privacy as can be guaranteed within the boundaries of lawful interception. We rely on a novel primitive, which we call Lawful-Interception Key Exchange (LIKE, in short), for which we formalize strong security requirements. As a second contribution, we describe an instantiation of LIKE using standard building blocks. We prove the security of our scheme in our model.

The LIKE primitive. In Lawful-Interception Key Exchange, after an initial setup and key-generation step, mobile users such as Alice and Bob can run an authenticated key-exchange protocol in the presence of their respective operators. This protocol allows (only) the end users to compute session keys, while the operators output a public value called a *session state*.

If the key computed by Alice and Bob is then subject to Lawful Interception, the authorities extract a trapdoor from the session state. Given all the trapdoors, the session key is reconstructed by an opening algorithm. Operators cannot recover Alice and Bob's key; their role is to verify the authenticity and well-formedness of the exchanged messages, and relay them if the verifications are successful. Else, the operators will abort and the session is halted.

We define the security of LIKE schemes in terms of three strong properties, presented below:

KS **Key-security:** If at least one authority and both users are honest for a given session, that session's key remains indistinguishable from a value picked at random from the key domain with respect to an adversary that can control all the remaining parties (including the other authorities and the operator);

NF **Non-frameability:** The collusion of malicious users, the authorities, and the operator cannot frame an honest user of participating to a session she has not been a party to;

HO **Honest operator:** The protocol is designed so that, if an honest operator forwards the session state of a session it deems correct, then the key recovered by the authorities is the one that the session transcript should have yielded. In other words, operators can prove that this protocol is compliant with LI specifications.

Our setup does not require us to trust either the authorities, or the operator. We provide two guarantees for mobile users (key-security and non-frameability), and one for the operator and authorities (honest operator). Intuitively, HO ensures both that LI will work correctly for the authorities, and that an honest operator can prove its protocol to be LI-compliant. On the other hand, KS and NF protect users from abuse, ensuring the protection of their privacy and the impossibility of their being framed by the authorities and operators. In particular, our protocol always protects users from the operator.

The last property –honest operator– is the most subtle. It states that two users cannot bypass lawful interception if they use keys yielded by this protocol. However, we note that our protocol does not – and cannot – ensure that Alice and Bob do, in fact, use only the keys provided by the protocol. Nothing prevents them from encrypting communications with a key they exchanged out-of-band, for instance. This is not a weakness of our protocol: indeed, Alice and Bob could have eluded LI in the same way by using the same out-of-band key to encrypt messages over a plaintext channel.

Our solution. As our second contribution, we present a LIKE scheme, in which we use bilinear pairings to allow the session key to be computed in two ways: either by the cooperation of a number of authorities, whose public keys are embedded on one side of the pairing, or by the two users Alice and Bob, whose contributions are embedded on the other side of the pairing.

This simple core idea, however, is insufficient to guarantee the strong properties we require. We allow authorities to be malicious, but this should not allow them to skew results for lawful interception; as a result, we require them to prove in zero-knowledge they have correctly generated their keys and trapdoors. Similarly, to ensure the HO property, the users use signatures of knowledge to prove to the operator that they have correctly generated their input. Finally, a user's presence or absence from a protocol should be unforgeable – therefore, each user will sign parts of the transcript to prove his or her presence.

We prove the security of our scheme based on the hardness of the Bilinear Decisional Diffie Hellman problem, the unforgeability of our signature scheme, and the security of both our proofs and signatures of knowledge.

Related work In mobile communications, the AKA secure-channel establishment protocol (introduced in 3G networks) is already meant to provide means of secure communication between the user and the operator. It is subject to LI requirements which persist through 4G technical specifications, to 5G documents and standards [1–3]. However, this protocol does not allow direct end-to-end-secure user-to-user communication. Our LIKE protocol provides much stronger privacy for users, even with respect to authorities and operators.

Lawful-interception key exchange is also strongly related to *key-escrow*: namely, a key-exchange scheme for which the key can be retrieved later by the authorities. Since the proposal of the Clipper chip in 1993[29] (proven to be flawed [10]), various key-escrow and key-recovery solutions were proposed [17], both using classical PKIs and in the context of identity-based cryptography. In our work, we strengthen user privacy beyond regular key-escrow and typical AKE-like key-security guarantees, considering the (to our best knowledge new) properties of non-frameability and honest operator.

PKI-based key-escrow schemes abound in the literature, in the context of Voice over IP [5], instant messaging [35], mobile communications [22], and generic secure-channel establishment [33, 7, 15, 25, 23, 24, 19, 27]. These schemes can be split into two categories, depending on whether the authorities reconstruct a master secret [5, 23, 24, 19] or a session-specific secret [33, 22, 35, 7, 15, 25, 27]. The former approach requires trust in the authorities performing LI, since the reconstructed master secret allows to open *all* key-exchange sessions, rather than only a few targeted ones.

The latter approach is closer to what we achieve here; in fact our goal of key-security is nicely described by Micali [27]. However, to the best of our knowledge, previously-existing schemes rely on mobile users (such as Alice and Bob) to send opening shares for their conversation dynamically to all the authorities. This massively increases complexity and decreases usability, since the authorities must always be online *and* store several elements per session (in view of possible ulterior LI). By contrast, our protocol allows authorities to remain offline until the moment of lawful interception, putting the burden of storing session states on the operator. This is a reasonable assumption, as it is what is currently done in mobile networks anyway.

Identity-based alternatives to the schemes described above were also proposed, including [34, 30, 16]. However, in identity-based schemes the key-generation center has access to the keys of the users. This makes users dependent on the honesty of the key-generation center, which defeats the purpose of enforcing strong law-enforcement requirements that protect user privacy.

Our work also marginally relates to lines of research that aim to limit mass-surveillance by making it difficult to recover keys [8, 36]. Our setup also resembles that of reverse firewalls [28] – which builds on related work pioneered by Young and Yung [37, 18, 21]. Although apparently similar, the setup of these works is complementary to ours. Kleptography describes ways for users

to abuse protocols such that the latter appear to be running normally, while in fact the permits subliminal information to leak. For instance in key-exchange, a government agency could want to substitute the implementation of the protocol by one that has a backdoor, to make the key recoverable even without formally doing LI. We do not address this problem here; instead, we treat the case in which Alice and Bob pick their input so as to make it hard for the authorities to retrieve the key *even when LI is correctly triggered* (the HO property).

2 Preliminaries

The notation $x \leftarrow y$ indicates that the variable x takes a value y and $x \xleftarrow{\$} X$ indicates that the variable x is chosen from the uniform distribution on X. We write $\mathsf{A}(x) \to a$ to express that the algorithm A, running on input x, outputs a, and $\mathsf{P}\langle \mathsf{A}(x), \mathsf{B}(y)\rangle(z) \to (a, b)$ to express that the protocol P implements the interactions of $\mathsf{A}(x) \to a$ and $\mathsf{B}(y) \to a$, where z is an additional public input of A and B. Let λ be a security parameter.

Definition 1 (BDDH assumption [11]). *Let $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T$ be groups of prime order p of length λ. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a type 3 bilinear map. The* Bilinear decisional Diffie-Hellman problem *(BDDH) assumption holds in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ if, given $(a, b, c, d_1) \xleftarrow{\$} (\mathbb{Z}_p^*)^4$, $d_0 \leftarrow abc$, and $\beta \xleftarrow{\$} \{0, 1\}$, no PPT adversary $\mathcal{A}$ can guess b from $(g_1^a, g_2^a, g_1^b, g_2^b, g_1^c, g_2^c, e(g_1, g_2)^{d_\beta})$ with non-negligible advantage. We denote by $\mathsf{Adv}^{\mathsf{BDDH}}(\lambda)$ the maximum advantage over all PPT adversaries.*

Definition 2 (Digital signatures). *A digital signature scheme $\mathsf{DS} = (\mathsf{SGen}, \mathsf{SSig}, \mathsf{SVer})$ is defined by three algorithms: $\mathsf{SGen}(1^\lambda) \to (\mathsf{PK}, \mathsf{SK})$; $\mathsf{SSig}(\mathsf{SK}, m) \to \sigma$, and $\mathsf{SVer}(\mathsf{PK}, m, \sigma) \to b$. Let Sig be an oracle that, on input m, returns $\mathsf{SSig}(\mathsf{SK}, m)$. Let $\mathcal{A}$ be PPT adversary. We define the following experiment:*

$\mathsf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A})$*:*
$(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{SGen}(1^\lambda)$*;* $(m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sig}(\cdot)}(\mathsf{PK})$*;*
Return 1 *if* (m, σ) *was not output by* $\mathsf{Sig}(\cdot)$ *and*
$\mathsf{SVer}(\mathsf{PK}, m, \sigma) = 1$*,* 0 *otherwise.*

A digital signature scheme DS *is* existentially unforgeable against chosen message attacks *(EUF-CMA) if, for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS},\mathcal{A}}(\lambda) = \mathbb{P}\left[\mathsf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS},\mathcal{A}}(\lambda) = 1\right]$ is negligible in λ. We denote by $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\lambda)$ the maximum advantage over all PPT adversaries.*

Signature of Knowledge. Let $\mathcal{R}$ be a binary relation and let $\mathcal{L}$ be a language such that $s \in \mathcal{L} \Leftrightarrow (\exists w, (s, w) \in \mathcal{R})$. A Non-Interactive Proof of Knowledge (NIPoK) [6] allows a prover to convince a verifier that he knows a witness w such that $(s, w) \in \mathcal{R}$. In this paper, we follow [12] and write $\mathsf{NIPoK}\{w : (w, s) \in \mathcal{R}\}$ for the proof of knowledge of w for the statement s and the relation $\mathcal{R}$. A *signature of knowledge* essentially allows one to sign a message and prove in zero-knowledge that a particular statement holds for the key [13]. In this paradigm, w is a secret key and s is the corresponding public key.

Definition 3 (Signature of Knowledge). *Let $\mathcal{R}$ be a binary relation and $\mathcal{L}$ be a language such that $s \in \mathcal{L} \Leftrightarrow (\exists w, (s, w) \in \mathcal{R})$. A* Signature of Knowledge *for $\mathcal{L}$ is a pair of algorithms* $(\mathsf{SoK}, \mathsf{SoKver})$ *with $\mathsf{SoK}_m\{w : (s, w) \in \mathcal{R}\} \to \pi$ and $\mathsf{SoKver}(m, s, \pi) \to b$, such that:*

– *Perfect Zero Knowledge: There exists a polynomial time algorithm* Sim*, the* simulator*, such that* $\mathsf{Sim}(m, s)$ *and* $\mathsf{SoK}_m\{w : (s, w) \in \mathcal{R}\}$ *follow the same probability distribution.*
– *Knowledge Extractor: There exists a PPT knowledge extractor* Ext *and a negligible function* ϵ_{SoK} *such that for any algorithm* $\mathcal{A}^{\mathsf{Sim}(\cdot,\cdot)}(\lambda)$ *having access to a simulator that forges signatures for chosen instance/message tuples and that outputs a fresh tuple* (s, π, m) *with* $\mathsf{SoKver}(m, s, \pi) = 1$*, the extractor* $\mathsf{Ext}^{\mathcal{A}}(\lambda)$ *outputs* w *such that* $(s, w) \in \mathcal{R}$ *having access to* $\mathcal{A}(\lambda)$ *with probability at least* $1 - \epsilon_{\mathsf{SoK}}(\lambda)$*.*

We omit to recall the definition of NIPoK which is the same as SoK without the messages.

3 LIKE protocols

We define lawful-interception (authenticated) key-exchange (LIKE) in terms of two mechanisms: a three-party AKE protocol featuring two mobile subscribers (which we call users) and a mobile network operator, and an Extract-and-Open mechanism involving a number of parties (which we call authorities).

Intuition. Our AKE component allows user Alice, subscribing to operator $\mathsf{O_A}$, and Bob, subscribing to $\mathsf{O_B}$, to compute a session key in the presence of those operators. However, $\mathsf{O_A}$ and $\mathsf{O_B}$ will not be able to compute the session key. Instead, they output some auxiliary material which we call *session state*, the latter allowing for session authentication and an ulterior key-recovery by a set of authorities.

In the Extract-and-Open component, each authority uses its secret key to *extract* a trapdoor from the operator's session state. Subsequently the trapdoors are used together to *open* the session key.

Formalization. Let USERS be a set of mobile users and OPS be a set of operators, such that each user is affiliated to precisely one operator. In particular, we equate users with uniquely affiliated hardware, such as SIM cards, rather than with people. We also consider a set of authorities AUTH of cardinality n, with elements indexed as $\Lambda_1, \ldots, \Lambda_n$.

Let $\mathsf{PARTIES}$ be the set of all participants: $\mathsf{USERS} \cup \mathsf{OPS} \cup \mathsf{AUTH}$. We require that mobile users have no super-role (they can be neither authorities, nor operators, *i.e.*, , $\mathsf{USERS} \cap \mathsf{AUTH} = \varnothing = \mathsf{USERS} \cap \mathsf{OPS}$). The case of operators being authorities is more complicated, and we describe it in more detail in Section 7. For now, assume that $\mathsf{OPS} \cap \mathsf{AUTH} = \varnothing$ as well.

Although we formally introduce parties, attributes, and oracles in the next section, we anticipate a few notations here. For each party P we use a dot notation to refer to *attributes* of that party (such as a long-term private or public key). For instance $\mathsf{A.PK}$ refers to Alice's public key and $\Lambda_i.\mathsf{SK}$ refers to the secret key of the i^{th} authority. We slightly abuse notation and write $\mathsf{O_A}$ to indicate Alice's operator – in reality, in our scheme we abstract the process of registering to an operator by allowing two parties to run the protocol in the presence of any two operators they choose to have. We also assume that Alice and Bob always interact with their respective operators during the protocol, which can be easily achieved by using, *e.g.*, the standard AKA protocol as an overlay of ours.

Definition 4. *A* lawful interception key exchange protocol *(*LIKE*) is defined by the following algorithms:*
– $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$*: Takes as input a security parameter in unary notation, and outputs* pp*, the public parameters of the system. They become known to all parties.*

- $\mathsf{UKeyGen}(\mathsf{pp}) \to (\mathsf{U.PK}, \mathsf{U.SK})$*: Takes as input the public parameters* pp *and outputs a* user key pair $(\mathsf{U.PK}, \mathsf{U.SK})$.
- $\mathsf{OKeyGen}(\mathsf{pp}) \to (\mathsf{O.PK}, \mathsf{O.SK})$*: Takes as input the public parameters* pp *and outputs the* operator key pair $(\mathsf{O.PK}, \mathsf{O.SK})$.
- $\mathsf{AKeyGen}(\mathsf{pp}) \to (\Lambda.\mathsf{PK}, \Lambda.\mathsf{SK})$*: Takes as input the public parameters* pp *and outputs an* authority key pair $(\Lambda.\mathsf{PK}, \Lambda.\mathsf{SK})$.
- $\mathsf{AKE}\langle \mathsf{A}(\mathsf{A.SK}), \mathsf{O_A}(\mathsf{O_A.SK}), \mathsf{O_B}(\mathsf{O_B.SK}), \mathsf{B}(\mathsf{B.SK})\rangle(\mathsf{PK_{A\to B}}) \to (\mathsf{k_A}, \mathsf{sst_A}, \mathsf{sst_B}, \mathsf{k_B})$*: An authenticated key-exchange protocol between two mobile subscribers* $(\mathsf{A}, \mathsf{B}) \in \mathsf{USERS}^2$*, their operators* $(\mathsf{O_A}, \mathsf{O_B}) \in \mathsf{OPS}^2$*, such that* $\mathsf{O_A}$ *and* $\mathsf{O_B}$ *provide active middleware for* A *and* B *during the protocol at all times (*A *and* B *never interact directly). Alice, Bob, and their operators each take as input their own secret key. In addition, each party has access to the set of public parameters* $\mathsf{PK_{A\to B}}$ *which contains: the public parameters* pp*, the public keys of Alice and Bob* $(\mathsf{A.PK}, \mathsf{B.PK})$*, and a vector of authority public keys* $\mathsf{APK} = (\Lambda_i.\mathsf{PK})_{i=1}^n$ *with* $\Lambda_i \in \mathsf{AUTH}$ *for all* i*. At the end of the protocol,* A *(resp.* B*) returns a* session secret key $\mathsf{k_A}$ *(resp.* $\mathsf{k_B}$*) and the operator* $\mathsf{O_A}$ *(resp.* $\mathsf{O_B}$*) returns a (public)* session state $\mathsf{sst_A}$ *(resp.* $\mathsf{sst_B}$*). In case of failure, the parties output a special symbol* $\perp$ *instead.*
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{sst}, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{O.PK}, \mathsf{APK}) \to \mathsf{b}$*: Takes as input a session state* sst*, two user public keys* $\mathsf{A.PK}$ *and* $\mathsf{B.PK}$*, the public key of an operator* $\mathsf{O.PK}$*, a set of public keys of the authorities* $\mathsf{APK} = (\Lambda_i.\mathsf{PK})_{i=1}^n$*, outputting a bit* $\mathsf{b} = 1$ *if the* sst *was correctly generated and authenticated by* O*, and* $\mathsf{b} = 0$ *otherwise.*
- $\mathsf{TDGen}(\mathsf{pp}, \Lambda.\mathsf{SK}, \mathsf{sst}) \to \Lambda.t$*: Takes as input an authority secret key* $\Lambda.\mathsf{SK}$ *and session state* sst*, and outputs a trapdoor* $\Lambda.t$.
- $\mathsf{Open}(\mathsf{pp}, \mathsf{sst}, \mathsf{APK}, \mathcal{T}) \to \mathsf{k}$*: Takes as input session state* sst*, a vector of public keys of the authorities* $\mathsf{APK} = (\Lambda_i.\mathsf{PK})_{i=1}^n$*, a vector of the corresponding trapdoors* $\mathcal{T} = (\Lambda_i.t)_{i=1}^n$ *, and outputs a session secret key* k*. In case of failure,* k *may take a special value* $\perp$.

Definition 5 (Correctness). *Let* λ *a security parameter and* n *an integer. Run* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{A.PK}, \mathsf{A.SK}) \leftarrow \mathsf{UKeyGen}(\mathsf{pp})$, $(\mathsf{B.PK}, \mathsf{B.SK}) \leftarrow \mathsf{UKeyGen}(\mathsf{pp})$, $(\mathsf{O_A.PK}, \mathsf{O_A.SK}) \leftarrow \mathsf{OKeyGen}(\mathsf{pp})$, $(\mathsf{O_B.PK}, \mathsf{O_B.SK}) \leftarrow \mathsf{OKeyGen}(\mathsf{pp})$. *For all* $i \in [\![1, n]\!]$, $(\Lambda_i.\mathsf{PK}, \Lambda_i.\mathsf{SK}) \leftarrow \mathsf{AKeyGen}(\mathsf{pp})$. *Let* $\mathsf{APK} \leftarrow (\Lambda_i.\mathsf{PK})_{i=1}^n$*. Then:*

- $\mathsf{PK_{A\to B}} \leftarrow (\mathsf{pp}, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{APK})$;
- $(\mathsf{k_A}, \mathsf{sst_A}, \mathsf{sst_B}, \mathsf{k_B}) \leftarrow \mathsf{AKE}\langle \mathsf{A}(\mathsf{A.SK}), \mathsf{O_A}(\mathsf{O_A.SK}), \mathsf{O_B}(\mathsf{O_B.SK}), \mathsf{B}(\mathsf{B.SK})\rangle(\mathsf{PK_{A\to B}})$;
- $\mathsf{b_A} \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{sst_A}, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{O_A.PK}, \mathsf{APK})$;
- *For all* i *in* $[\![1, n]\!]$, $\Lambda_i.t_\mathsf{A} \leftarrow \mathsf{TDGen}(\mathsf{pp}, \Lambda_i.\mathsf{SK}, \mathsf{sst_A})$;
- $\mathsf{k_A^*} \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{sst_A}, (\Lambda_i.\mathsf{PK})_{i=1}^n, (\Lambda_i.t_\mathsf{A})_{i=1}^n)$;
- $\mathsf{b_B} \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{sst_B}, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{O_B.PK}, \mathsf{APK})$;
- *For all* i *in* $[\![1, n]\!]$, $\Lambda_i.t_\mathsf{B} \leftarrow \mathsf{TDGen}(\mathsf{pp}, \Lambda_i.\mathsf{SK}, \mathsf{sst_B})$;
- $\mathsf{k_B^*} \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{sst_B}, \mathsf{APK}, (\Lambda_i.t_\mathsf{B})_{i=1}^n)$.

For any $(\mathsf{b_A}, \mathsf{b_B}, \mathsf{k_A}, \mathsf{k_A^*}, \mathsf{k_B}, \mathsf{k_B^*})$ *generated as above:* $\Pr[\mathsf{b_A} = \mathsf{b_B} = 1 \wedge \mathsf{k_A} = \mathsf{k_A^*} = \mathsf{k_B} = \mathsf{k_B^*}] = 1$.

To use a lawful-interception key-exchange scheme, users like Alice and Bob generate their keys using $\mathsf{UKeyGen}$, operators generate their keys using $\mathsf{OKeyGen}$, and each authority generates its keys using $\mathsf{AKeyGen}$. Then, Alice, Bob, and the operators run the protocol AKE using the vector of all the authority public keys. At the end of this protocol, Alice and Bob share a session secret key (not known by the operator) that they can use to communicate securely. Each operator returns a public session state sst, which is heavily protocol-dependent: its main purpose is to encapsulate

authenticated session data that will eventually allow the authorities to verify that the protocol was run correctly, and recover the session key. Finally, any party can check, by using the verification algorithm Verify, that the session secret key was honestly generated from a valid execution of the protocol between Alice, Bob and the operators.

The authorities may later retrieve this session secret key. Each authority uses its secret key and the session state to compute a trapdoor to that key. Before computing it, the authority checks the soundness of sst using Verify. By using all the trapdoors and the algorithm Open, one retrieves Alice and Bob's session key.

Depending on the lawful interception scenario, the Open algorithm may be run by one or multiple parties (we only require that whoever runs the algorithm possesses all the trapdoors). Our protocol is versatile and can adapt to any of these cases (see Section 7).

4 Security Model

We proceed to detail the adversarial model and formal security properties required for our new primitive LIKE. This is an essential contribution of our paper, for the following two reasons: first, because a formal treatment of such schemes allows us to *prove and quantify* the security of our scheme; and secondly, because the properties we formalize are much stronger than the obvious extension of the key-security properties required in regular authenticated key exchange. In particular, the properties of honest operator and non-frameability, for instance, are not achievable by schemes relying on a central key-distribution authority, like [34, 30, 16].

The remainder of this section is structured as follows. First we give the adversarial model, describing how parties and protocol instances are run. Then we list the oracles which adversaries can use to manipulate the honest parties. Finally, we describe formal security games for each of the given properties.

The adversarial model Each of the parties mentioned in Section 3 (the users, operators, and authorities) has a unique role, which is assumed not to change during the course of our security experiments. Each party P is associated with a number of attributes, listed below.

- (SK, PK): a tuple consisting of a long-term private key SK and a public key PK. These values may be instantiated to values output by UKeyGen, OKeyGen, or AKeyGen.
- γ: a corruption flag with a bit value, which indicates whether that party has been corrupted by the adversary or not. This bit is initially set to 0, but may be turned to 1 during the security game. Afterwards, the bit may no longer be changed to 0.

We continue to use the dot notation in Section 3; thus, *e.g.*, an operator's corrupt bit is denoted by $\mathsf{O}.\gamma$.

Alice, Bob, and their operators run the AKE in sessions. At each new protocol execution, a new *instance* of that party is created. Four party instances (one each for Alice, her operator, Bob's operator, and Bob himself) run the protocol together to form a session. We denote by π_{P}^i the i-th instance of party P.

Instances of each party automatically have access to the values of the long-term keys of those parties and their corruption bit. In addition, they keep track of the following attributes.

- sid: a session identifier consisting of a tuple of values (session specific), such as portions of the public parameters and the randomness. We stress that this attribute stores *only* the state

information pertinent to that protocol. This value is internally computable by the parties interacting in the protocol, but without involving any secret value. This attribute is instantiated to a default value $\perp$ and may change its value during the protocol's execution.
- PID: partner identifiers. If $\mathsf{P} \in \mathsf{USERS}$, then $\mathsf{PID} \in \mathsf{USERS}$ such that $\mathsf{P} \neq \mathsf{PID}$, else $\mathsf{P} \in \mathsf{OPS}$ and $\mathsf{PID} \in \mathsf{USERS}^2$ such that the two elements of the vector PID are different.
- OID: operator identifiers. If $\mathsf{P} \in \mathsf{USERS}$ then $\mathsf{OID} \in \mathsf{OPS}^2$ such that $|\mathsf{OID}| = 2$ (the operators are distinct). Else $\mathsf{OID} \in \mathsf{OPS}$.
- AID: authority identifiers such that $\mathsf{AID} \in \mathsf{AUTH}^n$.
- α: the instance's accept flag, which is undefined ($\alpha = \perp$) until the instance terminates its AKE execution. If the instance terminates by aborting the protocol, it sets $\alpha = 0$, else if the protocol terminates without error, it sets $\alpha = 1$.
- k: the instance's session key, first initialized to a special value $\perp$. If the protocol terminates without error, k takes the value returned by the protocol. An operator instance does not have this attribute (it is replaced by the session state attribute sst defined below).
- sst: a set of variables storing additional state of the instance's protocol execution. If the protocol terminates without error, sst takes the value returned by the protocol. User instances do not have this attribute.
- ρ: the instance's reveal bit, first initialized to 0 and set to 1 if the adversary reveals a session key $\mathsf{k} \neq \perp$. Operator instances do not have this attribute.
- b: a bit chosen uniformly at random upon the creation of the instance.
- τ: the transcript of the ongoing session, first initialized as $\perp$, turning to the ordered list of messages sent and received by that instance in the order they are sent and received.

We define an auxiliary function $\mathsf{IdentifySession}(\mathsf{sst}, \pi)$ that takes as input a session state sst and a party instance π, and outputs 1 if π took part in the session where sst was created, and 0 otherwise. This is because during the session opening, the authorities must be able to extract and verify the session identifier for themselves.

Notice that, unlike in typical authenticated key-exchange protocols, we have two different kind of parties in this protocol (users and operators), and three types of parties that partner an instance (as indicated by the attributes PID, OID, and AID). The instance stores mobile user partners in PID, operator partners in OID, and authorities partnering it in AID.

In addition, notice that $\mathsf{IdentifySession}$ and sst are protocol dependent, *i.e.* they will have a different instantiation depending on the protocol we analyse.

For our AKE component we require the notion of matching conversation defined as follows:

Definition 6 (Matching instances). *For any* $(i, j) \in \mathbb{N}^2$ *and* $(\mathsf{A}, \mathsf{B}) \in \mathsf{USERS}^2$ *such that* $A \neq B$*, we say that* π_{A}^i *and* π_{B}^j have matching conversation *if all the following conditions hold:* $\pi_{\mathsf{A}}^i.\mathsf{sid} \neq \perp$*,* $\pi_{\mathsf{A}}^i.\mathsf{sid} = \pi_{\mathsf{B}}^j.\mathsf{sid}$*, and* $\pi_{\mathsf{A}}^i.\mathsf{AID} = \pi_{\mathsf{B}}^j.\mathsf{AID}$*. If two instances* π_{A}^i *and* π_{B}^j *have matching conversation, we sometimes say, by abuse of language, that* π_{A}^i matches π_{B}^j*.*

Oracles. We define the security properties of our novel LIKE primitive in terms of games that an adversary plays against a challenger (simulating all the honest parties). In order to manipulate the environment, the adversary will have access to some or all of the oracles presented below. Intuitively, we give the adversary the ability to register honest or malicious participants (the $\mathsf{Register}$ oracle), initiate new sessions (the $\mathsf{NewSession}$ oracle), interact in the AKE protocol (the Send oracle), corrupt parties (the $\mathsf{Corrupt}$ oracle), reveal session keys (the Reveal oracle), or reveal trapdoors towards

potentially opening the key by the LI process (the $\mathsf{RevealTD}$ oracle). Finally, we use a testing oracle (Test), which will test whether the session key remains indistinguishable from random from the adversary's point of view.

For each oracle, we add that they abort if they receive a query that is not well formatted, or if the challenger does not have enough information to correctly answer the query.

- $\mathsf{Register}(\mathsf{P}, \mathtt{role}, \mathsf{PK}) \to \bot \cup \mathsf{P.PK}$: On input a party identity $\mathsf{P} \notin \mathsf{USERS} \cup \mathsf{OPS} \cup \mathsf{AUTH}$, a role $\mathtt{role} \in \{\mathtt{user}, \mathtt{operator}, \mathtt{authority}\}$ and a public key PK:
  - If $\mathtt{role} = \mathtt{user}$ (resp. $\mathtt{operator}$, $\mathtt{authority}$), the oracle adds P to the set USERS (resp. OPS, AUTH).
  - If $\mathtt{role} = \mathtt{user}$ (resp. $\mathtt{operator}$ and $\mathtt{authority}$) and $\mathsf{PK} = \bot$, then it runs $\mathsf{UKeyGen}(\mathsf{pp}) \to (\mathsf{P.PK}, \mathsf{P.SK})$ (resp. $\mathsf{OKeyGen}$ and $\mathsf{AKeyGen}$).
  - If $\mathsf{PK} \neq \bot$, it sets the corruption bit γ to 1 for this party, sets $\mathsf{P.PK} = \mathsf{PK}$ and $\mathsf{P.SK} = \bot$.

  Finally, it returns $\mathsf{P.PK}$.
- $\mathsf{NewSession}(\mathsf{P}, \mathsf{PID}, \mathsf{OID}, \mathsf{AID}) \to \pi^i_{\mathsf{P}}$: On input a party $\mathsf{P} \in \mathsf{USERS} \cup \mathsf{OPS}$, if $\mathsf{P} \in \mathsf{USERS}$ then $\mathsf{PID}, \mathsf{OID}$ and AID must verify that $\mathsf{PID} \in \mathsf{USERS}$ such that $\mathsf{P} \neq \mathsf{PID}$, $\mathsf{OID} \in \mathsf{OPS}^2$ and $\mathsf{AID} \in \mathsf{AUTH}^*$ such that $|\mathsf{AID}| \neq 0$. If $\mathsf{P} \in \mathsf{OPS}$ then $\mathsf{PID}, \mathsf{OID}$ and AID verify that $\mathsf{PID} \in \mathsf{USERS}^2$ such that PID contains two different users, $\mathsf{OID} \in \mathsf{OPS}$ and $\mathsf{AID} \in \mathsf{AUTH}^*$ such that $|\mathsf{AID}| \neq 0$. On the i^{th} call to this oracle, it returns a new instance π^i_{P} with the attributes PID, OID and AID.
- $\mathsf{Send}(\pi^i_{\mathsf{P}}, m) \to m'$: Sends the message m to the instance π^i_{P} and returns a new message m', as specified by the protocol AKE. This message can be a special value $\bot$ in case of failure (if the instance does not exist, has already rejected the session, the session is finished, the protocol aborts, m is not well formatted, or $\mathsf{P.SK} = \bot$).
- $\mathsf{Reveal}(\pi^i_{\mathsf{P}}) \to \mathsf{k}$: For an instance of $\mathsf{P} \in \mathsf{USERS}$ that has accepted the session ($\alpha = 1$), it returns the session key $\pi^i_{\mathsf{P}}.\mathsf{k}$ and sets the reveal bit $\pi^i_{\mathsf{P}}.\rho$ to 1. For an instance of $\mathsf{P} \in \mathsf{OPS}$ that has accepted the session ($\alpha = 1$), it returns the session state $\pi^i_{\mathsf{P}}.\mathsf{sst}$ and sets the reveal bit $\pi^i_{\mathsf{P}}.\rho$ to 1. If the session has not been accepted ($\alpha \neq 1$), the oracle returns $\bot$.
- $\mathsf{Corrupt}(\mathsf{P}) \to \mathsf{P.SK}$: It returns the SK of the party $\mathsf{P} \in \mathsf{USERS} \cup \mathsf{OPS} \cup \mathsf{AUTH}$ and sets the corruption bit $\mathsf{P}.\gamma$ to 1 for this party and any of its instances.
- $\mathsf{Test}(\pi^i_{\mathsf{P}}) \to \widetilde{\mathsf{k}}$: Can be queried for an instance of $\mathsf{P} \in \mathsf{USERS}$ that has accepted the session ($\alpha = 1$). If $\pi^i_{\mathsf{P}}.\mathsf{b} = 0$, it returns the session key $\pi^i_{\mathsf{P}}.\mathsf{k}$. Otherwise, it returns a randomly sampled value r from the same domain as $\pi^i_{\mathsf{P}}.\mathsf{k}$. If the instance has not accepted the session, it returns $\bot$. If this oracle had been previously queried and had returned a value different from $\bot$, it will return $\bot$ (this oracle can effectively be queried only once).
- $\mathsf{RevealTD}(\mathsf{sst}, \mathsf{A}, \mathsf{B}, \mathsf{O}, (\Lambda_i)_{i=1}^n, l) \to \Lambda_l.t$: If $\mathsf{Verify}(\mathsf{pp}, \mathsf{sst}, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{O.PK}, (\Lambda_i.\mathsf{PK})_{i=1}^n) = 1$, then it runs $\Lambda_l.t \leftarrow \mathsf{TDGen}(\mathsf{pp}, \Lambda_l.\mathsf{SK}, \mathsf{sst})$ and returns $\Lambda_l.t$, else it returns $\bot$.

We emphasize that the operators involved in the session do not contribute to the security of the key in the traditional AKE sense. Instead, operators verify the soundness of the exchanges and produce a session state sst; this value then serves to prove to the authorities that the operator outputting correctly arbitrated the protocol from its point of view. We do not require the operators to agree on sst, and perform the opening procedure on a single operator at a time (since this is what would happen in most real-life situations). If one operator validates, but the other operator aborts the protocol, then ultimately the session will not take place.

Security games We define the security for LIKE protocols in terms of three properties: key-security (KS), non-frameability (NF), and honest operator (HO).

$\mathsf{Exp}^{\mathsf{KS}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$:
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$;
$\mathcal{O}_{\mathsf{KS}} \leftarrow \left\{ \begin{matrix} \mathsf{Register}(\cdot,\cdot,\cdot), \mathsf{NewSession}(\cdot,\cdot,\cdot), \mathsf{Send}(\cdot,\cdot), \\ \mathsf{Reveal}(\cdot), \mathsf{RevealTD}(\cdot,\cdot), \mathsf{Corrupt}(\cdot,\cdot), \mathsf{Test}(\cdot) \end{matrix} \right\}$;
$(i, \mathsf{P}, \mathsf{d}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KS}}}(\lambda, \mathsf{pp})$;
If $\pi^i_{\mathsf{P}}.\mathsf{k}$ is fresh and $\pi^i_{\mathsf{P}}.\mathsf{b} = \mathsf{d}$, then return 1;
Else $b' \xleftarrow{\$} \{0,1\}$, return b'.

Table 1. The key-security (KS) experiment.

Key-security. Our KS game is an extension of the standard notion of AKE security [9]. The adversary has access to all the oracles presented above (subject to the definition of key-freshness defined below). Its goal is to distinguish the real key used by Alice and Bob from a key picked randomly from the same domain.

We give the adversary considerably more power than in traditional 2-party AKE games. The attacker can adaptively corrupt all but the two users targeted in the challenge, all the operators, and all but one of the authorities. The adversary may also register illegitimate users and learn all but one of the trapdoors involved in any session.

In other words, this definition only allows the key to be known by the parties having computed it (Alice and Bob), and by the collusion of *all* the authorities (LI). We rule out these two circumstances by the following notion of key freshness. All other (collusions of) parties should fail to distinguish the real session key from a random one.

Definition 7 (Key freshness). *Let π^j_{P} be the j-*th *instance of a party $\mathsf{P} \in \mathsf{USERS}$ and denote by $\mathcal{A}$ a PPT adversary against a* LIKE *scheme. We parse $\pi^j_{\mathsf{P}}.\mathsf{PID}$ as P' and $\pi^j_{\mathsf{P}}.\mathsf{AID}$ as $(\Lambda_i)_{i=1}^n$. We say that $\pi^j_{\mathsf{P}}.\mathsf{k}$ is* fresh *if all the following conditions hold:*

- *P has accepted in session j ($\pi^j_{\mathsf{P}}.\alpha = 1$), P remained uncorrupted ($\mathsf{P}.\gamma = 0$) up to the moment when $\pi^j_{\mathsf{P}}.\alpha$ became 1, and the session key remains unrevealed ($\pi^j_{\mathsf{P}}.\rho = 0$).*
- *if π^j_{P} matches some $\pi^k_{\mathsf{P}'}$ for $k \in \mathbb{N}$, then P' has accepted in session k ($\pi^k_{\mathsf{P}'}.\alpha = 1$), P' was uncorrupted ($\mathsf{P}'.\gamma = 0$) up to when $\pi^k_{\mathsf{P}'}.\alpha$ became 1, and the session key remains unrevealed ($\pi^k_{\mathsf{P}'}.\rho = 0$).*
- *if no $\pi^k_{\mathsf{P}'}$ matches π^j_{P}, P' is uncorrupted ($\mathsf{P}'.\gamma = 0$).*
- *there exists $l \in [\![1, n]\!]$ such that $\mathcal{A}$ has never queried* RevealTD *on input $(\mathsf{sst}, \mathsf{A}, \mathsf{B}, (\Lambda'_i)_{i=1}^{n'}, l')$ such that:*
  - *Λ_l is uncorrupted ($\Lambda_l.\gamma = 0$) and $\Lambda_l = \Lambda'_{l'}$;*
  - *$\mathsf{IdentifySession}(\mathsf{sst}, \pi^j_{\mathsf{P}}) = 1$.*

In this game, the adversary eventually outputs an instance for which it guesses its test bit b. This instance must be key-fresh with respect to Definition 7. More formally, we consider the following key-security game:

Definition 8. *We define the advantage of an adversary $\mathcal{A}$ in the $\mathsf{Exp}^{\mathsf{KS}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$ experiment in Table 1 as:*

$$\mathsf{Adv}^{\mathsf{KS}}_{\mathsf{LIKE},\mathcal{A}}(\lambda) := \left| \mathbb{P}\left[\mathsf{Exp}^{\mathsf{KS}}_{\mathsf{LIKE},\mathcal{A}}(\lambda) = 1 \right] - \frac{1}{2} \right|.$$

A lawful interception authenticated key-exchange scheme LIKE *is* key-secure *if for all PPT adversaries $\mathcal{A}$ the advantage $\mathsf{Adv}^{\mathsf{KS}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$ is negligible as a function of the security parameter λ.*

$\underline{\mathsf{Exp}^{\mathsf{NF}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)}$:
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$;
$\mathcal{O}_{\mathsf{NF}} \leftarrow \left\{ \begin{matrix} \mathsf{Register}(\cdot,\cdot,\cdot), \mathsf{NewSession}(\cdot,\cdot,\cdot), \mathsf{Send}(\cdot,\cdot), \\ \mathsf{Reveal}(\cdot), \mathsf{RevealTD}(\cdot,\cdot), \mathsf{Corrupt}(\cdot,\cdot) \end{matrix} \right\}$;
$(\mathsf{sst}, \mathsf{P}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{NF}}}(\lambda, \mathsf{pp})$;
If $\exists\ (\mathsf{A},\mathsf{B}) \in \mathsf{USERS}^2$, $n \in \mathbb{N}$, $\mathsf{O} \in \mathsf{OPS}$ and $(\Lambda_i)_{i=1}^n \in \mathsf{AUTH}^n$ s.t.:
  $\mathsf{Verify}(\mathsf{pp}, \mathsf{sst}, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{O.PK}, (\Lambda_i.\mathsf{PK})_{i=1}^n) = 1$ and
  $\mathsf{P} \in \{\mathsf{A},\mathsf{B}\}$ and
  $\mathsf{P}.\gamma = 0$ and
  $\forall i$, if $\pi^i_{\mathsf{P}} \neq \perp$ then $\mathsf{IdentifySession}(\mathsf{sst}, \pi^i_{\mathsf{P}}) = 0$ or $\pi^i_{\mathsf{P}}.\alpha = 0$,
Then return 1,
Else return 0.

Table 2. The non-frameability (NF) experiment.

$\underline{\mathsf{Exp}^{\mathsf{HO}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)}$:
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$;
$\mathcal{O}_{\mathsf{HO}} \leftarrow \left\{ \begin{matrix} \mathsf{Register}(\cdot,\cdot,\cdot), \mathsf{NewSession}(\cdot,\cdot,\cdot), \mathsf{Send}(\cdot,\cdot), \\ \mathsf{Reveal}(\cdot), \mathsf{RevealTD}(\cdot,\cdot), \mathsf{Corrupt}(\cdot,\cdot) \end{matrix} \right\}$;
$(j, \mathsf{sst}, \mathsf{A}, \mathsf{B}, \mathsf{O}, (\Lambda_i, \Lambda_i.t)_{i=1}^n) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{HO}}}(\lambda, \mathsf{pp})$;
If $\mathsf{O}.\gamma = 1$ then return $\perp$;
If $\mathsf{Verify}(\mathsf{pp}, \mathsf{sst}, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{O.PK}, (\Lambda_i.\mathsf{PK})_{i=1}^n) = 0$ then return $\perp$;
If $\mathsf{IdentifySession}(\mathsf{sst}, \pi^j_{\mathsf{O}}.\mathsf{sid}) = 0$ then return $\perp$;
$k_* \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{sst}, (\Lambda_i.\mathsf{PK})_{i=1}^n, (\Lambda_i.t)_{i=1}^n)$;
Return $(k_*, \pi^j_{\mathsf{O}}, \{\mathsf{P}_i.\mathsf{PK}\}_{i=1}^{q_r})$).

Table 3. The honest-operator HO experiment. Here, q_r denotes the number of queries to the $\mathsf{Register}$ oracle, and P_i denotes the party input as the i-th such query.

Non-Frameability. In this game, the adversary attempts to frame a target user P^* of taking part in an AKE session (corresponding to session state sst) in which P^* never took part or which P^* never accepted. The adversary is allowed to corrupt all parties in the system apart from P^*. More formally, we consider the following security experiment.

Definition 9. *The advantage of an adversary $\mathcal{A}$ in the non-frameability experiment $\mathsf{Exp}^{\mathsf{NF}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$ in Table 2 is defined as:*

$$\mathsf{Adv}^{\mathsf{NF}}_{\mathsf{LIKE},\mathcal{A}}(\lambda) = \mathbb{P}\left[\mathsf{Exp}^{\mathsf{NF}}_{\mathsf{LIKE},\mathcal{A}}(\lambda) = 1\right].$$

A lawful interception authenticated key-exchange scheme LIKE *is* non-frameable *if for all PPT adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{NF}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$ is negligible as a function of the security parameter λ.*

Honest Operator. The HO game captures the fact that honest operators have the power of aborting sessions if the messages exchanged are not well-formed or authentic. The adversary aims to forge a valid session state for which the operators have not aborted, but for which the authorities would Open to another key than that computed by Alice and Bob. The attacker can corrupt all the parties except for the operator which is assumed to eventually approve the session state, and it can even provide trapdoors.

Ideally, we would like LIKE schemes to guarantee that if the (honest) operator deems a session to be correctly run, then the extracted key (output by the experiment in Table 3) will be the one used by Alice and Bob. However, malicious participants Alice and Bob could run a protocol perfectly and then use a different key (which they might exchange out of band) to encrypt messages. This is inherent to secure-channel establishment when the participants are malicious. The best an LIKE

protocol can guarantee is that the extracted key is the one that *would have resulted* from an honest run of the protocol whose session state is given. We express this in terms of a *key extractor*.

Intuitively, given as input an operator instance and a set of public keys, the extractor outputs the key k that should have been output by the partnered instances running that session, if they were honest.

Definition 10 (Key extractor). *For any* LIKE*, a key extractor* $\mathsf{Extract}(\cdot,\cdot)$ *is a deterministic unbounded algorithm such that, for any users* A *and* B*, operators* $\mathsf{O_A}$ *and* $\mathsf{O_B}$*, and set of* n *authorities* $(\Lambda_i)_{i=1}^n$*, any set* $\{\mathsf{pp},\ \mathsf{A.PK},\ \mathsf{A.SK}, \mathsf{B.PK},\ \mathsf{B.SK},\ \mathsf{O_A.PK},\ \mathsf{O_A.SK},\ \mathsf{O_B.PK},\ \mathsf{O_B.SK},\ \mathsf{k},\ \mathsf{sst},\ \mathsf{APK} = (\Lambda_i.\mathsf{PK})_{i=1}^n,\ (\Lambda_i.\mathsf{SK})_{i=1}^n,\ \tau_\mathsf{A}, \tau_\mathsf{B}\}$ *generated as follows:*

$\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$*;* $(\mathsf{A.PK}, \mathsf{A.SK}) \leftarrow \mathsf{UKeyGen}(\mathsf{pp})$*;* $(\mathsf{B.PK}, \mathsf{B.SK}) \leftarrow \mathsf{UKeyGen}(\mathsf{pp})$*;*

$(\mathsf{O_A.PK}, \mathsf{O_A.SK}) \leftarrow \mathsf{OKeyGen}(\mathsf{pp})$*;*

$(\mathsf{O_B.PK}, \mathsf{O_B.SK}) \leftarrow \mathsf{OKeyGen}(\mathsf{pp})$*;*

For all $i \in [\![1, n]\!]$*,* $(\Lambda_i.\mathsf{PK}, \Lambda_i.\mathsf{SK}) \leftarrow \mathsf{AKeyGen}(\mathsf{pp})$*;*

$(\mathsf{k}, \mathsf{sst_A}, \mathsf{sst_B}, \mathsf{k}) \leftarrow \mathsf{AKE}\langle \mathsf{A}(\mathsf{A.SK}), \mathsf{O_A}(\mathsf{O_A.SK}), \mathsf{O_B}(\mathsf{O_B.SK}), \mathsf{B}(\mathsf{B.SK})\rangle(\mathsf{pp}, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{APK})$*;*

τ_A *is the transcript of the execution yielding* $\mathsf{sst_A}$ *from* $\mathsf{O_A}$*'s point of view;*

τ_B *is the transcript of the execution yielding* $\mathsf{sst_B}$ *from* $\mathsf{O_B}$*'s point of view;*

it holds that:

- $\{\mathsf{O_A.PK}, \mathsf{O_B.PK}, \mathsf{A.PK}, \mathsf{B.PK}\} \cup \{\Lambda_i.\mathsf{PK}\}_{i=1}^n \subseteq \mathsf{PPK}$*,*
- *for any oracle instance* $\pi_{\mathsf{O_A}}$ *such that* $\pi_{\mathsf{O_A}}.\tau = \tau_\mathsf{A}$*,* $\pi_{\mathsf{O_A}}.\mathsf{PID} = (\mathsf{A}, \mathsf{B})$*,* $\pi_{\mathsf{O_A}}.\mathsf{AID} = (\Lambda_i)_{i=1}^n$*, and* $\pi_{\mathsf{O_A}}.\mathsf{sst} = \mathsf{sst}$*, we have:* $\Pr[\mathsf{Extract}(\pi_{\mathsf{O_A}}, \mathsf{PPK}) = \mathsf{k}] = 1$*.*
- *for any oracle instance* $\pi_{\mathsf{O_B}}$ *such that* $\pi_{\mathsf{O_B}}.\tau = \tau_\mathsf{B}$*,* $\pi_{\mathsf{O_B}}.\mathsf{PID} = (\mathsf{A}, \mathsf{B})$*,* $\pi_{\mathsf{O_B}}.\mathsf{AID} = (\Lambda_i)_{i=1}^n$*, and* $\pi_{\mathsf{O_B}}.\mathsf{sst} = \mathsf{sst}$*, we have:* $\Pr[\mathsf{Extract}(\pi_{\mathsf{O_B}}, \mathsf{PPK}) = \mathsf{k}] = 1$*.*

We stress the fact that our extractor is unbounded – indeed it must be so, or otherwise the property of key-security would be violated (the extractor would allow the operator to find the session key).

Definition 11. *For any lawful interception key-exchange scheme* LIKE *that admits a key extractor* Extract *the advantage of an adversary* $\mathcal{A}$ *in the honest-operator game* $\mathsf{Exp}^{\mathsf{HO}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$ *in Table 3 is defined as:*

$$\mathsf{Adv}^{\mathsf{HO}}_{\mathsf{LIKE},\mathcal{A}}(\lambda) = \mathbb{P}\left[(k_*, \pi_\mathsf{O}, \mathsf{PPK}) \leftarrow \mathsf{Exp}^{\mathsf{HO}}_{\mathsf{LIKE},\mathcal{A}}(\lambda); k \leftarrow \mathsf{Extract}(\pi_\mathsf{O}, \mathsf{PPK}) : \begin{array}{l} k \neq \perp \wedge\ \ k_* \neq \perp \\ \wedge\ k \neq k_* \end{array}\right].$$

A LIKE *scheme is* honest-operator secure *if, for all adversaries running in time polynomial in* λ*, the value* $\mathsf{Adv}^{\mathsf{HO}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$ *is negligible as a function of* λ*.*

5 Our protocol

To instantiate our LIKE scheme, we will need the following primitives (for which some background is given in Section 2):

- A signature scheme $\mathsf{DS} = (\mathsf{SGen}, \mathsf{SSig}, \mathsf{SVer})$;
- A non-interactive zero-knowledge proof of knowledge $(\mathsf{NIPoK}, \mathsf{NIPoKver})$ for a cyclic group $\mathbb{G}_1 = \langle g_1 \rangle$ allowing to prove knowledge of the discrete logarithm of a value $y = g_1^x$ for some private witness x. We denote this as $\mathsf{NIPoK}\,\{x : y = g_1^x\}$.

– A signature of knowledge scheme $(\mathsf{SoK}, \mathsf{SoKver})$ allowing to prove the knowledge of a discrete logarithm in a second cyclic group $\mathbb{G}_2 = \langle g_2 \rangle$.
– A non-interactive zero-knowledge proof of knowledge $(\mathsf{NIPoK}, \mathsf{NIPoKver})$ for two groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_T = \langle g_T \rangle$ of same order p allowing to prove knowledge of the discrete logarithm of two values $y_1 = g_1^x$ and $y_T = g_T^x$ for some private witness x. We denote this by $\mathsf{NIPoK}\ \{x : y_1 = g_1^x \wedge y_T = g_T^x\}$.

The proof and the signature of knowledge of a discrete logarithm can be instantiated with Schnorr's protocol [32]. This protocol consists in three passes: a commitment, a challenge and a response. The Fiat-Shamir heuristic allows to change Schnorr's protocol into a non-interactive proof by using the hash of the statement concatenated with the commitment as challenge, and into a signature of knowledge by adding the message in this hash [13]. The proof of the discrete logarithm equality can be instantiated with The Fiat-Shamir heuristic applied on the protocol of Chaum and Pedersen [14].

Our scheme follows the syntax in Section 3. We divide its presentation in four main components: (a) setup and key generation, (b) authenticated key-exchange, (c) public verification, and (d) lawful interception.

Setup and key generation. This part instantiates the four following algorithms of the LIKE syntax presented in Section 3.

– $\mathsf{Setup}(1^\lambda)$: Based on λ, choose $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, and $\mathbb{G}_T$, three groups of prime order p of length λ, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ a type 2 bilinear mapping, and output $\mathsf{pp} = (1^\lambda, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p, g_1, g_2)$.
– $\mathsf{UKeyGen}(\mathsf{pp})$: Run $(\mathsf{U.PK}, \mathsf{U.SK}) \leftarrow \mathsf{SGen}(1^\lambda)$ and return $(\mathsf{U.PK}, \mathsf{U.SK})$.
– $\mathsf{OKeyGen}(\mathsf{pp})$: Run $(\mathsf{O.PK}, \mathsf{O.SK}) \leftarrow \mathsf{SGen}(1^\lambda)$ and return $(\mathsf{O.PK}, \mathsf{O.SK})$.
– $\mathsf{AKeyGen}(\mathsf{pp})$: Pick $\Lambda.\mathsf{SK} \xleftarrow{\$} \mathbb{Z}_p^*$, compute $\Lambda.\mathsf{pk} \leftarrow g_1^{\Lambda.\mathsf{SK}}$ and $\Lambda.\mathsf{ni} \leftarrow \mathsf{NIPoK}\left\{\Lambda.\mathsf{SK} : \Lambda.\mathsf{pk} = g_1^{\Lambda.\mathsf{SK}}\right\}$, set $\Lambda.\mathsf{PK} \leftarrow (\Lambda.\mathsf{pk}, \Lambda.\mathsf{ni})$ and return $(\Lambda.\mathsf{PK}, \Lambda.\mathsf{SK})$.

The setup algorithm is run only once, to generate the groups required to instantiate our bilinear pairing. The users and operators generate signature keys, to be used with our digital-signature scheme DS in the authenticated key-exchange step. Authorities not only generate a private and public key-pair, but have to prove (in zero-knowledge) that they know the private key. This prevents attacks in which one or multiple authorities try to "annihilate" the contribution of one or more of the authorities and break key security. We explain the role of this proof in more detail after presenting the rest of our scheme. All key-generation algorithms are run only once per party.

Authenticated key exchange. Every time two users communicate, they run the the AKE component of our LIKE syntax, $\mathsf{AKE}\langle \mathsf{A}(\mathsf{A.SK}), \mathsf{O_A}(\mathsf{O_A.SK}), \mathsf{O_B}(\mathsf{O_B.SK}), \mathsf{B}(\mathsf{B.SK}) \rangle\ (\mathsf{pp}, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{APK})$.

We depict the protocol in more detail in Figure 2. For readability we abbreviate the picture and "merge" $\mathsf{O_A}$ and $\mathsf{O_B}$. Notice particularly that the operators' roles are very similar to one another, and neither $\mathsf{O_A}$, nor $\mathsf{O_B}$ will learn the keys established by the end points.

In Figure 2, all the parties involved in the protocol (namely Alice, Bob, and their respective operators) will verify the public keys of the n authorities and abort if their associated NIZK proofs of knowledge are not correct. Clearly, however, if one user (say Alice) has already performed this step for a given authority, it can, in practice, skip the verification of its zero-knowledge proof during a specific future session.

The heart of the protocol is the exchange between Alice and Bob. Alice generates a secret x and sends $X_1 = g_1^x, X_2 = g_2^x$ to Bob, with an associated signature of knowledge linking this key share to ω. Bob proceeds in the same manner, sampling a secret y and sending to Alice $Y = g_2^y$ and

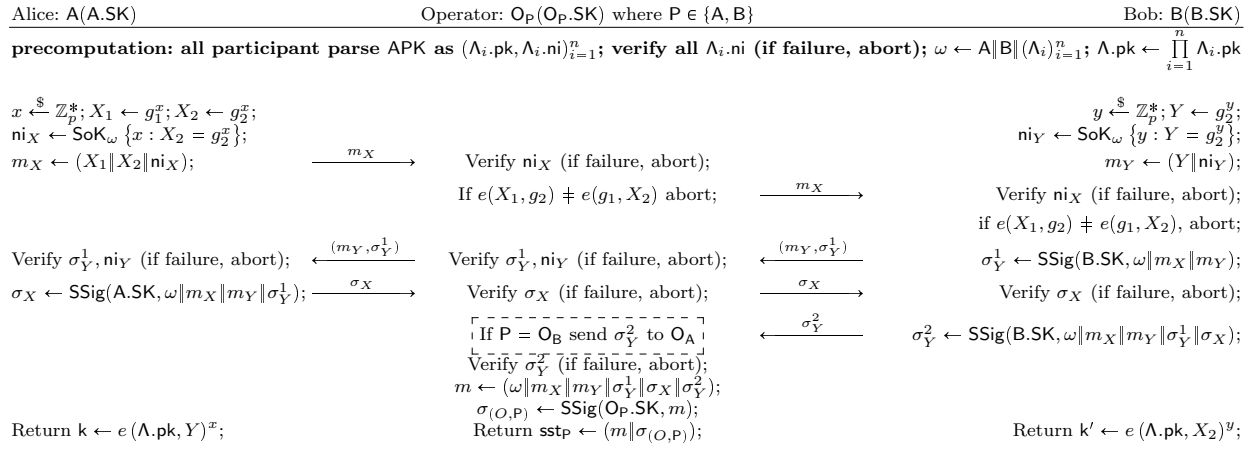

Fig. 2. The AKE component of our LIKE construction, with both operators $\mathsf{O_A}$ and $\mathsf{O_B}$ under a single heading (for readability). Unless specified, each operator runs in turn each line of the protocol, then forwards the message to the next participant or else aborts. Exceptionally, the only message *not* forwarded $\mathsf{O_A}$ to Alice is marked in the dashed box.

a signature of knowledge. The two parties also send to each other a signature over the transcript, thus authenticating each other and verifying the integrity of the conversation.

The two operators sit in between the two users. They verify the messages that they receive; if the message is correct, they forward it. If not, they abort the protocol. An exception is Bob's last message, which is verified by both operators, but does not reach Alice. The two operators check the signatures of knowledge, the digital signatures and the fact that the two values sent by Alice share the same discrete logarithm (by doing one pairing computation).

Given the public value Y, the public keys of the authorities, and her private exponent x, Alice computes her session key as a bilinear map applied on the product of all the public keys of the authorities and Bob's value Y, all raised to her secret x: $\mathsf{k} = e(\prod_{i=1}^{n} \Lambda_i.\mathsf{pk}, Y)^x$. Due to bilinearity, this is equal to $e(\prod_{i=1}^{n} \Lambda_i.\mathsf{pk}, g_2^y)^x = e(\prod_{i=1}^{n} \Lambda_i.\mathsf{pk}, g_2^x)^y = e(\prod_{i=1}^{n} \Lambda_i.\mathsf{pk}, X_2)^y$, which is what Bob computes on his end. This indicates the correctness of our scheme with respect to the endpoints.

The operator's output is the session state sst, the signed session transcript from the their point of view.

Verification. We instantiate Verify as follows:

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{sst}, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{O.PK}, \mathsf{APK}) \to b$: Parse APK as a set $(\Lambda_i.\mathsf{PK})_{i=1}^n$ and parse each $\Lambda_i.\mathsf{PK}$ as $(\Lambda_i.\mathsf{pk}, \Lambda_i.\mathsf{ni})$, set $\omega \leftarrow \mathsf{A}\|\mathsf{B}\|(\Lambda_i)_{i=1}^n$. Parse sst as $\omega'\|m_X\|m_Y\|\sigma_Y^1\|\sigma_X\|\sigma_Y^2\|\sigma_O$, m_X as $X_1\|X_2\|\mathsf{ni}_X$ and m_Y as $Y\|\mathsf{ni}_Y$. if:
  - For all $i \in [\![1, n]\!]$, $\mathsf{NIPoKver}(\Lambda_i.\mathsf{pk}, \Lambda_i.\mathsf{ni}) = 1$;
  - $e(X_1, g_2) = e(g_1, X_2)$;
  - $\mathsf{SoKver}(\omega, (g_2, X_2), \mathsf{ni}_X) = 1$;
  - $\mathsf{SoKver}(\omega, (g_2, Y), \mathsf{ni}_Y) = 1$;
  - $\mathsf{SVer}(\mathsf{B.PK}, \sigma_Y^1, \omega\|m_X\|m_Y) = 1$;
  - $\mathsf{SVer}(\mathsf{A.PK}, \sigma_X, \omega\|m_X\|m_Y\|\sigma_Y^1) = 1$;
  - $\mathsf{SVer}(\mathsf{B.PK}, \sigma_Y^2, \omega\|m_X\|m_Y\|\sigma_Y^1\|\sigma_X) = 1$;
  - $\mathsf{SVer}(\mathsf{O.PK}, \sigma_O, \omega\|m_X\|m_Y\|\sigma_Y^1\|\sigma_X\|\sigma_Y^2) = 1$;

  then the algorithm returns 1, else it returns 0.

Intuitively, one verifies that Alice and Bob participated in the session by verifying their signatures. The same can be said about the operator that output sst (which could be either $\mathsf{O_A}$ or $\mathsf{O_B}$), whose final signature is verifiable with its public key. In addition, the verification algorithm performs the checks required by the protocol from the operator: verifying the proofs presented by the authorities, checking that X_1 and X_2 contain the same discrete logarithm ($e(X_1, g_2) = e(g_1, X_2)$), and verifying all the signatures of knowledge.

Lawful Interception. As presented in Section 3, lawful interception for our protocol follows an extract-and-open strategy. We employ two algorithms, one run by each authority in order to generate a trapdoor, and the other run by one or a multitude of parties, cumulatively in possession of all the trapdoors for a given session.

- $\mathsf{TDGen}(\mathsf{pp}, \Lambda.\mathsf{SK}, \mathsf{sst})$: Parse sst as $(\omega \| m_X \| m_Y \| \sigma_Y^1 \| \sigma_X \ \| \sigma_Y^2 \| \sigma_O)$. Compute $\Lambda.t_1 \leftarrow e(X_1, Y)^{\Lambda.\mathsf{SK}}$, $\Lambda.t_2 \leftarrow \mathsf{NIPoK}\left\{\Lambda.\mathsf{SK} : \Lambda.\mathsf{PK} = g_1^{\Lambda.\mathsf{SK}} \wedge \Lambda.t_1 = e(X_1, Y)^{\Lambda.\mathsf{SK}}\right\}$ and $\Lambda.t \leftarrow (\Lambda.t_1, \Lambda.t_2)$, and returns $\Lambda.t$.
- $\mathsf{Open}(\mathsf{pp}, \mathsf{sst}, \mathsf{APK}, \mathcal{T})$: Parse $\mathcal{T}$ as $(\Lambda_i.t)_{i=1}^n$, parse sst as $\mathsf{A} \| \mathsf{B} \| (\Lambda_i)_{i=1}^n \| m_X \| m_Y \| \sigma_Y^1 \| \sigma_X \| \sigma_Y^2 \| \sigma_O$, m_X as $X_1 \| X_2 \| \mathsf{ni}_X$ and m_Y as $Y \| \mathsf{ni}_Y$, APK as $(\Lambda_i.pk)_{i=1}^n$, parse each $\Lambda_i.\mathsf{PK}$ as $(\Lambda_i.\mathsf{pk}, \Lambda_i.\mathsf{ni})$, each $\Lambda_i.t$ as $(\Lambda_i.t_1, \Lambda_i.t_2)$ and verify the non-interactive proof of knowledge with the corresponding algorithm: $\mathsf{NIPoKver}((g_1, \Lambda_i.\mathsf{pk}, (X_1, Y), \Lambda_i.t_1), \Lambda_i.t_2)$; if any verification fails, the Open algorithm returns $\perp$. Compute $\mathsf{k} \leftarrow \prod_{i=1}^n (\Lambda_i.t_1)$ and return it.

Informally, each authority Λ_i generates as trapdoor the value $\Lambda_i.t_1 = e(X_1, Y)^{\Lambda_i.\mathsf{SK}} = e(g_1, g_2^{xy})$. In order to recover the session key (by means of the Open algorithm), all the trapdoors from $\Lambda_1.t_1$ to $\Lambda_n.t_1$ are multiplied. We obtain $\prod_{i=1}^n \Lambda_1.t_i = \prod_{i=1}^n e(g_1^x, g_2^y)^{\Lambda_i.\mathsf{SK}} = e(g_1, g_2^{xy})^{\sum_{i=1}^n \Lambda_i.\mathsf{SK}} = e(g_1^{\sum_{i=1}^n \Lambda_i.\mathsf{SK}}, g_2^{x \cdot y}) = e(\prod_{i=1}^n \Lambda_i.\mathsf{PK}, X_2)^y$. We now recognize the key as computed by Bob. Our scheme is correct with respect to LI.

Complexity. In our key exchange protocol, each party runs in linear time in the number of authorities. This complexity is due to the verification of the zero-knowledge proof for each authority public key, and the computation of the product of these public keys. We stress that the parties can pre-computed the proofs verifications and the product, moreover, while the parties run the protocol for the same authority set, they have not to recompute this, meaning that in practice, our protocol is in constant time most of the time. Finally, we note that the trapdoor generation algorithm also runs in constant time.

6 Proofs

We show that the LIKE scheme we propose in Section 5 has the non-frameability, key-security and honest operator properties defined in Section 4. We provide proof sketches, followed by the full proofs. However, we first provide an insight into the role of some of our less-obvious building blocks, namely the proofs and signatures of knowledge.

Insight: proofs and signatures of knowledge. During the protocol, Alice and Bob use the two signatures of knowledge on the public parameters of the session (denoted ω): $\mathsf{SoK}_\omega\{x : X_2 = g_2^x\}$ and $\mathsf{SoK}_\omega\{y : Y = g_2^y\}$. In the absence of these proofs, the adversary could recover the key of an honest session through the following attack. Let X_1, X_2, Y be the values honestly generated by the users in the targeted session. Choose two random values r and s, run the protocol using $X_1' \leftarrow X_1^r$, $X_2' \leftarrow X_2^r$ and $Y' \leftarrow Y^s$ for two corrupted users and the same set of authorities, obtain the corresponding sst, and run the $\mathsf{RevealTD}$ oracle on sst for each authority. Using the algorithm

Open, the adversary obtains: $\mathsf{k}' = \prod_{i=1}^{n}(X_1', Y_2')^{\Lambda_i.\mathsf{SK}} = \left(\prod_{i=1}^{n}(X_1, Y_2)^{\Lambda_i.\mathsf{SK}}\right)^{r\cdot s}$. It can then compute the targeted key as $\mathsf{k} = (\mathsf{k}')^{1/r\cdot s}$. However, this attack cannot be done by the adversary if it must prove its knowledge of the discrete logarithm of X_2' and Y'. Moreover, it cannot reuse X_2 and Y together with the signature of the honest user, since that signature uses the identity of the users as a message.

Each authority public key $\Lambda.\mathsf{pk}$ must be coupled with a proof of knowledge of the corresponding secret key $\mathsf{NIPoK}\left\{\Lambda.\mathsf{SK} : \Lambda.\mathsf{pk} = g_1^{\Lambda.\mathsf{SK}}\right\}$. Without this proof, an authority Λ_j in the set $(\Lambda_i)_{i=1}^n$ could pick a random r and compute: $\Lambda_j.\mathsf{pk} = g_1^r / \left(\prod_{i=1;i\neq j}^{n} \Lambda_i.\mathsf{pk}\right)$. In this case, the secret key computed by Alice will be: $\mathsf{k} = e\left(\prod_{i=1}^{n} \Lambda_i.\mathsf{pk}, Y\right)^x = e(g_1^r, g_2^y)^x = e(X_1, Y)^r$. Then the dishonest authority can discover the key by computing $e(X_1, Y)^r$. This attack is not possible if the authority must prove the knowledge of its secret key.

Finally, the proof $\mathsf{NIPoK}\{\Lambda.\mathsf{SK} : \Lambda.\mathsf{PK} = g_1^{\Lambda.\mathsf{SK}} \wedge \Lambda.t_1 = e(X_1, Y)^{\Lambda.\mathsf{SK}}\}$, which is a part of the trapdoor generated by an authority, ensures that the trapdoor has been correctly formed. This is essential for the honest-operator property: without this proof, the authority can output a fake trapdoor, thus distorting the output of the algorithm Open. **Proofs.** We proceed with the formal security statements and proofs. In the following, we define $\mathsf{sid} := X_2 \| Y$ (see Figure 2).

We define the algorithm $\mathsf{IdentifySession}(\mathsf{sst}, \pi_{\mathsf{P}}^j)$ for some party P and integer j as follows. Parsing sst as:

$$\mathsf{A}\|\mathsf{B}\|(\Lambda_i)_{i=1}^n\|X_1\|X_2\|\mathsf{ni}_X\|Y\|\mathsf{ni}_Y\|\sigma_Y^1\|\sigma_X\|\sigma_Y^2\|\sigma_O,$$

the algorithm $\mathsf{IdentifySession}(\mathsf{sst}, \pi_{\mathsf{P}}^j)$ returns 1 iff:

- $X_2\|Y = \pi_{\mathsf{P}}^i.\mathsf{sid}$,
- if π_{P}^j plays the role of Alice then $\mathsf{P} = \mathsf{A}$ and $\pi_{\mathsf{P}}^j.\mathsf{PID} = \mathsf{B}$, else $\pi_{\mathsf{P}}^j.\mathsf{PID} = \mathsf{A}$ and $\mathsf{P} = \mathsf{B}$, and
- $\pi_{\mathsf{P}}^j.\mathsf{AID} = (\Lambda_i)_{i=1}^n$.

Theorem 1. *If our protocol is instantiated with an EUF-CMA signature scheme* DS*, and with extractable and zero-knowledge proofs/signature of knowledge, and if the BDDH assumption holds, then our protocol is key-secure. Moreover, for all PPT adversaries* $\mathcal{A}$ *doing at most* q_{r} *queries to the oracle* $\mathsf{Register}$*,* q_{ns} *queries to the oracle* $\mathsf{NewSession}$*,* q_{s} *queries to the oracle* Send *and* q_{t} *queries to the oracle* $\mathsf{RevealTD}$*, we have:*

$$\mathsf{Adv}^{\mathsf{KS}}_{\mathsf{LIKE},\mathcal{A}}(\lambda) \leqslant \frac{q_{\mathsf{s}}^2}{p} + q_{\mathsf{ns}} \cdot q_{\mathsf{r}}^2 \cdot \Big(\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\lambda) + q_{\mathsf{ns}} \cdot q_{\mathsf{r}} \cdot \Big((2 \cdot q_{\mathsf{t}} + q_{\mathsf{s}}) \cdot \epsilon_{\mathsf{SoK}}(\lambda) + q_{\mathsf{r}} \cdot \epsilon_{\mathsf{NIPoK}}(\lambda) + \mathsf{Adv}^{\mathsf{BDDH}}(\lambda)\Big)\Big).$$

Proof sketch. We begin by proving that the adversary has a negligible probability of winning the key-security experiment by querying the oracle Test on an instance that matches no other instance. Notably, if the tested instance does not abort the protocol, the adversary will have to break the EUF-CMA of the signature scheme to generate the expected signatures without using a matching session.

Thus, the targeted instance must have a matching one. By key-freshness, $\mathcal{A}$ must test a key generated by two honest users, such that the trapdoor of at least one honest authority has never been queried to the oracle $\mathsf{RevealTD}$. We prove (by a reduction) that $\mathcal{A}$ can only win by breaking

the BDDH assumption. Let $(W_*, X_*, Y_*, W'_*, X'_*, Y'_*, Z_*)$ be a BDDH instance. We set W_* as the part of the public key $\Lambda.\mathsf{pk}$ of the honest authority, and we set X_2 as X'_*, X_1 as X_* and Y as Y'_* for the session that matches the tested instance. Then, we build the key as follows, where Λ is the honest authority: $\mathsf{k} \leftarrow Z_* \prod_{i=1;\Lambda_i \neq \Lambda}^{n} e(X_*, Y'_*)^{\Lambda_i.\mathsf{SK}}$. To compute the secret keys of the authorities controlled by the adversary, we run the extractor on the proofs of knowledge of the discrete logarithm of the public keys $\Lambda_i.\mathsf{PK}$. If Z_* is a random value, k will be random for the adversary, else $Z_* = e(X_*, Y'_*)^{\Lambda.\mathsf{SK}}$. Moreover, we simulate the oracle $\mathsf{RevealTD}$ on sessions with values X and Y chosen by the adversary by using the extractor on the signatures of knowledge of their discrete logarithms.

Proof. **(Th.1: Key-security).** We will show that $\mathsf{Adv}^{\mathsf{KS}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$ is negligible for any PPT adversary $\mathcal{A}$, and thus our LIKE scheme is key-secure, by the following sequence of games:
Game G_0: This game is the same as $\mathsf{Exp}^{\mathsf{KS}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$.
Game G_1: This game is similar to G_0, but aborts if the Send oracle returns twice the same element as X_2 or Y. An abort only happens if two out of the q_s queried instances choose the same randomness from $\mathbb{G}_2$ (which is of size p), yielding:

$$|\mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_0] - \mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_1]| \leqslant q_\mathsf{s}^2/p.$$

Let $\pi^{i*}_{\mathsf{P}_*}$ denote a tested instance. Excluding collisions for X_2 and Y implies that $\pi^{i*}_{\mathsf{P}_*}$ now has at most one matching instance. Indeed, suppose two or more instances matching $\pi^{i*}_{\mathsf{P}_*}$ exist. We parse $\pi^{i*}_{\mathsf{P}_*}.\mathsf{sid}$ as $Z_0 \| Z_1$ where Z_i (for $i \in \{0,1\}$) was generated by $\pi^{i*}_{\mathsf{P}_*}$.

By Def. 6, all instances matching $\pi^{i*}_{\mathsf{P}_*}$ must sample the same $Z_{1-i} \in \mathbb{G}_2$ – impossible after G_1. .
Game G_2: Let P_i be the i-th party instantiated by $\mathsf{Register}$. Game G_2 proceeds as G_1 except that it begins by choosing $(u, v, w) \xleftarrow{\$} [\![1, q_\mathsf{ns}]\!] \times [\![1, q_\mathsf{r}]\!]^2$. If the adversary returns $(i_*, \mathsf{P}_*, \mathsf{d}_*)$ such that, given $\mathsf{P}'_* \leftarrow \pi^{i*}_{\mathsf{P}_*}.\mathsf{PID}$, we have $i_* \neq u$ or $\mathsf{P}_* \neq \mathsf{P}_v$ or $\mathsf{P}'_* \neq \mathsf{P}_w$, then G_2 aborts, returning a random bit. The adversary increases its winning advantage by a factor equalling the probability of guessing correctly:

$$|\mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_1] - 1/2| \leqslant q_\mathsf{ns} \cdot q_\mathsf{r}^2 |\mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_2] - 1/2|.$$

Game G_3: Let $(i_*, \mathsf{P}_*, \mathsf{d}_*)$ be the adversary's test session, guessed by G_2. Let $\mathsf{P}'_* \leftarrow \pi^{i*}_{\mathsf{P}_*}.\mathsf{PID}$. Game G_3 works as G_2, except that, if there exists no $\pi^{k}_{\mathsf{P}'_*}$ matching $\pi^{i*}_{\mathsf{P}_*}$, the experiment aborts and returns a random bit. We claim that for any adversary $\mathcal{A}$:

$$|\mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_2] - \mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_3]| \leqslant \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\lambda).$$

Assume to the contrary that there exists an adversary $\mathcal{A}$ that wins G_2 with probability $\epsilon_\mathcal{A}(\lambda)$ by returning a guess $(i_*, \mathsf{P}_*, \mathsf{d}_*)$ such that, setting $\mathsf{P}'_* \leftarrow \pi^{i*}_{\mathsf{P}_*}.\mathsf{PID}$, no $k \in \mathbb{N}$ exists such that $\pi^{i*}_{\mathsf{P}_*}$ and $\pi^{k}_{\mathsf{P}'_*}$ match. Game G_2 demands $\mathsf{P}'_* \leftarrow \mathsf{P}_w$ (guessed by G_2); key-freshness (Def. 7) requires P_w to be uncorrupted and ending in an accepting state. We use $\mathcal{A}$ to build a PPT adversary $\mathcal{B}$ that breaks the $\mathsf{EUF\text{-}CMA}$ security of DS with non-negligible probability. $\mathcal{B}$ receives the verification key $\hat{\mathsf{PK}}$, initializes $\mathcal{L}_S \leftarrow \varnothing$, and faithfully simulates G_2 to $\mathcal{A}$, except for $\mathcal{A}$'s following queries:
Oracle $\mathsf{Register}(\mathsf{P}, \mathtt{role}, \mathsf{PK})$**:** If $\mathsf{P} = \mathsf{P}_w$ with $\mathsf{P.PK} = \perp$, then $\mathcal{B}$ sets $\mathsf{P.PK} \leftarrow \hat{\mathsf{PK}}$.
Oracle $\mathsf{Send}(\pi^i_\mathsf{P}, m)$**:** There are two particular cases: $\mathsf{P} = \mathsf{P}_w$ and $\mathsf{P} = \mathsf{P}_*$. If $\mathsf{P} = \mathsf{P}_w$, then $\mathcal{B}$ queries its $\mathsf{Sig}(\cdot)$ oracle to answer $\mathcal{A}$'s queries. Depending on the role of P_w and the protocol step, $\mathcal{B}$ runs one of: $\sigma^1_Y \leftarrow \mathsf{Sig}(\omega \| m_X \| m_Y)$, $\sigma_X \leftarrow \mathsf{Sig}(\omega \| m_X \| m_Y \| \sigma^1_Y)$, or $\sigma^2_Y \leftarrow \mathsf{Sig}(\omega \| m_X \| m_Y \| \sigma^1_Y \| \sigma_X)$. Here,

if P_w is the initiator, $\omega = \mathsf{P}_w \| \mathsf{P}_w.\mathsf{PID} \| (\Lambda_l)_{l=1}^n$; else $\omega = \mathsf{P}_w.\mathsf{PID} \| \mathsf{P}_w \| (\Lambda_l)_{l=1}^n$. Moreover, $m_X = X_1 \| X_2 \| \mathsf{ni}_X$ and $m_Y = Y \| \mathsf{ni}_Y$. The message/signature pairs are stored in $\mathcal{L}_S$. Since $\mathsf{sid} = X_2 \| Y$, the elements X_2 and Y, the identities P_w and $\pi^i_{\mathsf{P}_w}.\mathsf{PID}$, and the set of authorities $\pi^i_{\mathsf{P}_w}.\mathsf{AID} = (\Lambda_l)_{l=1}^n$ are parts of the message signed in σ_X, σ_Y^1 and σ_Y^2.

If $\mathsf{P} = \mathsf{P}_*$, $i = i_*$, and $\pi^i_{\mathsf{P}}.\mathsf{PID} = \mathsf{P}_w$, if $\mathsf{SVer}(\mathsf{P}_w.\mathsf{PK}, \sigma_Y^2, M_Y) = 1$ for $M_Y \leftarrow \omega \| m_X \| m_Y \| \sigma_Y^1 \| \sigma_X$, and $(M_Y, \sigma_Y^2) \notin \mathcal{L}_S$, $\mathcal{B}$ aborts the game and returns (M_Y, σ_Y^2); else, if $\mathsf{SVer}(\mathsf{P}_w.\mathsf{PK}, \sigma_X, M_X) = 1$ for $M_X \leftarrow \omega \| m_X \| m_Y \| \sigma_Y^1$ and $(M_X, \sigma_X) \notin \mathcal{L}_S$, $\mathcal{B}$ aborts the game and returns (M_X, σ_X).

Oracle $\mathsf{Corrupt}(\mathsf{P})$**:** If $\mathsf{P} = \mathsf{P}_w$, $\mathcal{B}$ aborts (due to G_2).

$\mathcal{B}$ wins if it sends its challenger a message/signature pair $(M, \sigma) \notin \mathcal{L}_S$ such that $\mathsf{SVer}(\hat{\mathsf{PK}}, \sigma, M) = 1$ with $\hat{\mathsf{PK}} = \mathsf{P}_w.\mathsf{PK}$. We first argue that $\mathcal{A}$ must query Send on input $\mathsf{P} = \mathsf{P}_*$, $i = i_*$, and $\pi^i_{\mathsf{P}}.\mathsf{PID} = \mathsf{P}_w$, on message $M_Y = \omega \| m_X \| m_Y \| \sigma_Y^1 \| \sigma_X$ such that $\mathsf{SVer}(\mathsf{P}_w.\mathsf{PK}, \sigma_Y^2, M_Y) = 1$, or on message $M_X \leftarrow \omega \| m_X \| m_Y \| \sigma_Y^1$ such that $\mathsf{SVer}(\mathsf{P}_w.\mathsf{PK}, \sigma_X, M_X) = 1$. Indeed, if $\mathcal{A}$ does not, the (honest) target instance $\pi^{i_*}_{\mathsf{P}_*}$ rejects.

Now we can assume that $\mathcal{A}$ has queried Send either with σ_X or with σ_Y as above. We have two cases: the submitted message/signature pair is in $\mathcal{L}_S$, or it is not. If the latter happens, clearly $\mathcal{B}$ wins. Assume that the former happens, *i.e.*, the signature σ_Y^2 or σ_X are in $\mathcal{L}_S$ (generated by $\mathcal{B}$'s oracle). We recall that by assumption $\mathcal{A}$'s challenge instance has no matching instance, *i.e.*, there exists no $\pi^j_{\mathsf{P}_w}$ such that $\pi^j_{\mathsf{P}_w}.\mathsf{sid} \neq \perp$ or $\pi^j_{\mathsf{P}_w}.\mathsf{sid} = \pi^{i_*}_{\mathsf{P}_*}.\mathsf{sid}$, and $\pi^j_{\mathsf{P}_w}.\mathsf{AID} = \pi^{i_*}_{\mathsf{P}_*}.\mathsf{AID}$. However, if $\pi^j_{\mathsf{P}_w}.\mathsf{sid} \neq \pi^{i_*}_{\mathsf{P}_*}.\mathsf{sid}$, or $\pi^j_{\mathsf{P}_w}.\mathsf{AID} \neq \pi^{i_*}_{\mathsf{P}_*}.\mathsf{AID}$, then the signature is generated for the fresh message and $\mathcal{A}$ would win.

Thus, $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS},\mathcal{B}}(\lambda) = \epsilon_{\mathcal{A}}(\lambda)$, concluding the proof.

After G_2, G_3, either a unique instance $\pi^r_{\mathsf{P}_w}$ exists, matching $\pi^{i_*}_{\mathsf{P}_*}$ or the experiment returns a random bit.

Game G_4: Game G_4 runs as G_3 except that it begins by picking $r \xleftarrow{\$} [\![1, q_{\mathsf{ns}}]\!]$ (a guess for the matching instance). If $\mathcal{A}$ returns $(i_*, \mathsf{P}_*, \mathsf{d}_*)$ such that $\pi^{i_*}_{\mathsf{P}_*}$ and $\pi^r_{\mathsf{P}_w}$ do not match, then the experiment returns a random bit. The advantage of $\mathcal{A}$ on G_4 increases w.r.t. that in G_3 by a factor equalling the correct guessing probability :

$$|\mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_3] - 1/2| \leqslant q_{\mathsf{ns}} |\mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_4] - 1/2|.$$

Game G_5: Game G_5 proceeds as G_4, except that it begins by picking $l \xleftarrow{\$} [\![1, q_{\mathsf{r}}]\!]$. If the l-th party queried to the oracle $\mathsf{Register}$ is not authority, or, if it is an authority (we will denote it Λ_l), and is either corrupted, or $\mathsf{RevealTD}$ is called on the query $(\mathsf{sst}, \mathsf{A}, \mathsf{B}, (\Lambda'_i)_{i=1}^{n'}, l')$ such that $\Lambda_l = \Lambda'_{l'}$ and $\mathsf{IdentifySession}(\mathsf{sst}, \pi^u_{\mathsf{P}_v}.\mathsf{sid}) = 1$, then the experiment aborts by returning a random bit. By key-freshness (Def. 7), if no index l exists such that Λ_l is uncorrupted ($\Lambda_l.\gamma = 0$) and $\mathsf{RevealTD}$ has never been called on the query $(\mathsf{sst}, \mathsf{A}, \mathsf{B}, (\Lambda'_i)_{i=1}^{n'}, l')$ such that $\Lambda_l = \Lambda'_{l'}$ and $\mathsf{IdentifySession}(\mathsf{sst}, \pi^j_{\mathsf{P}}.\mathsf{sid}) = 1$, then the experiment returns a random bit. Thus, the advantage of $\mathcal{A}$ in G_5 is superior to that in G_4 by a factor equalling the guessing probability:

$$|\mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_4] - 1/2| \leqslant q_{\mathsf{ns}} |\mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_5] - 1/2|.$$

Game G_6: Let Ext denote the knowledge extractor of the signature of knowledge. This game is the same as G_5 except that it begins by initializing $\mathcal{L}[\,] \leftarrow \varnothing$ and:

- each time that the Send oracle generates an element $d \xleftarrow{\$} \mathbb{Z}_p^*$ and $D \leftarrow g_2^d$ together with a signature of knowledge $\mathsf{ni}_D \leftarrow \mathsf{SoK}_\omega\left\{d : D = g_2^d\right\}$, it sets $\mathcal{L}[(D, \omega, \sigma_D)] \leftarrow d$;

– each time that the oracles Send or $\mathsf{RevealTD}$ verify a valid signature of knowledge $\mathsf{SoKver}(\omega, (g_2, D), \mathsf{ni}_D) = 1$ in a query sending by $\mathcal{A}$ such that $\mathcal{L}[(D, \omega, \sigma_D)] = \perp$, it runs the key extractor $\mathsf{Ext}(\lambda)$ on $\mathcal{A}$ in order to extract the witness d that matches the proof ni_D. If $g_2^d \neq D$ then the experiment aborts by returning a random bit, else it sets $\mathcal{L}[(D, \omega, \sigma_D)] \leftarrow d$.

The difference between G_5 and G_6 is the possibility of the extractor failing when it is called. Since $\mathsf{RevealTD}$ requires 2 calls (for the verification of sst) and Send, one at each query, it holds that:

$$|\mathbb{P}[\mathcal{A} \text{ wins } G_5] - \mathbb{P}[\mathcal{A} \text{ wins } G_6]| \leqslant (2 \cdot q_{\mathsf{t}} + q_{\mathsf{s}}) \cdot \epsilon_{\mathsf{SoK}}(\lambda).$$

Game G_7: Let Ext denote the extractor of the NIZK proof of knowledge $\mathsf{NIPoK}\left\{d : D = g_1^d\right\}$. Game G_7 runs as G_6, except it begins by initializing an empty list $\mathcal{L}'[\,] \leftarrow \varnothing$ and:

– Honest authority: if $\mathsf{Register}$ generates $(\Lambda.\mathsf{PK}, \Lambda.\mathsf{SK})$ for an authority Λ, it sets $\mathcal{L}'[\Lambda.\mathsf{PK}] \leftarrow \Lambda.\mathsf{SK}$;
– Malicious authority: if $\mathsf{Register}$ receives a query $(\Lambda, \mathtt{role}, \mathsf{PK})$ with $\mathtt{role} = \mathtt{authority}$ and $\mathsf{PK} \neq \perp$, it sets $\mathsf{PK} \leftarrow \Lambda.\mathsf{PK}$ and parses PK as $(\Lambda.\mathsf{pk}, \Lambda.\mathsf{ni})$. If $\mathsf{NIPoKver}((g_1, \Lambda.\mathsf{pk}), \Lambda.\mathsf{ni}) = 1$, then G_7 runs the extractor $\mathsf{Ext}(\lambda)$ on $\mathcal{A}$ to extract the witness $\Lambda.\mathsf{SK}$ for $\Lambda.\mathsf{ni}$. If $g_1^{\Lambda.\mathsf{SK}} \neq \Lambda.\mathsf{pk}$ then the experiment aborts by returning a random bit, else it sets $\mathcal{L}'[\Lambda.\mathsf{PK}] \leftarrow \Lambda.\mathsf{SK}$.

Once more, the difference between the games is the possibility that Ext fails in at least one of the calls to the registration oracle, yielding:

$$|\mathbb{P}[\mathcal{A} \text{ wins } G_6] - \mathbb{P}[\mathcal{A} \text{ wins } G_7]| \leqslant q_{\mathsf{r}} \cdot \epsilon_{\mathsf{NIPoK}}(\lambda).$$

Finally, we claim that:

$$|\mathbb{P}[\mathcal{A} \text{ wins } G_7] - 1/2| \leqslant \mathsf{Adv}_{\mathcal{A}}^{\mathsf{BDDH}}(\lambda).$$

We prove this claim by reduction. Assume that $\mathcal{A}$ wins G_7 with non-negligible probability $\epsilon_{\mathcal{A}}(\lambda)$. We show how to build an algorithm $\mathcal{B}$ that breaks the BDDH problem with probability $\epsilon_{\mathcal{A}}(\lambda)$.

In what follows, $\mathsf{Sim}_{\mathsf{NIPoK}}$ denotes the simulator of the proofs of knowledge and $\mathsf{Sim}_{\mathsf{SoK}}$ is the simulator of the signature of knowledge. For readability, if the context is clear, we use the same notation for the simulators of the two proof of knowledge systems we use to instantiate our scheme.

$\mathcal{B}$ plays the BDDH in the setting $(\mathbb{G}_1 = \langle g_1 \rangle, \mathbb{G}_2 = \langle g_2 \rangle, \mathbb{G}_T, e, p)$ and receives $(\hat{W}, \hat{X}, \hat{Y}, \hat{W}', \hat{X}', \hat{Y}', \hat{Z})$ from its challenger. Then $\mathcal{B}$ sets $\mathsf{pp} \leftarrow (1^\lambda, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p, g_1, g_2)$, it runs $\mathcal{A}(\mathsf{pp})$ and simulates G_7 to $\mathcal{A}$ as in the real game except for following cases.

Oracle $\mathsf{Register}(\mathsf{P}, \mathtt{role}, \mathsf{PK}) \rightarrow \mathsf{P}.\mathsf{PK}$**:** On the l-th query, if $\mathtt{role} \neq \mathtt{authority}$ or $\mathsf{PK} \neq \perp$, $\mathcal{B}$ aborts and returns a random bit (this is a faithful simulation since G_5), else it sets $\Lambda_l \leftarrow \mathsf{P}$; $\Lambda_l.\mathsf{pk} \leftarrow \hat{W}$; $\Lambda_l.\mathsf{ni} \leftarrow \mathsf{Sim}_{\mathsf{NIPoK}}(g_1, \hat{W})$; $\Lambda_l.\mathsf{PK} \leftarrow (\Lambda_l.\mathsf{pk}, \Lambda_l.\mathsf{ni})$ and returns $\Lambda_l.\mathsf{PK}$.

Oracle $\mathsf{Send}(\pi_{\mathsf{P}}^i, m)$**:** If $\mathsf{P} = \mathsf{P}_*$ and $i = i_*$ (challenge instance, guessed in G_3), or if $\mathsf{P} = \mathsf{P}_w$ and $i = r$ (unique matching instance, guessed in G_4), then we distinguish two cases:

– P plays the role of Alice: then $\mathcal{B}$ proceeds as in G_7 except for generating $(X_1, X_2, \mathsf{ni}_X)$ by setting $X_2 \leftarrow \hat{X}'$, $X_1 \leftarrow \hat{X}$ and by running $\mathsf{ni}_X \leftarrow \mathsf{Sim}_{\mathsf{SoK}}(\omega, (g_2, \hat{X}'))$, where $\omega = (\mathsf{P}_v \| \mathsf{P}_w \| \pi_{\mathsf{P}_v}^i.\mathsf{AID})$.
– P plays the role of Bob: then $\mathcal{B}$ proceeds as in G_7 except for generating (Y, ni_Y) by setting $Y \leftarrow \hat{Y}'$ and by running $\mathsf{ni}_Y \leftarrow \mathsf{Sim}_{\mathsf{SoK}}(\omega, (g_2, \hat{Y}'))$, where $\omega = (\mathsf{P}_w \| \mathsf{P}_v \| \pi_{\mathsf{P}_w}^i.\mathsf{AID})$.

At the end of the protocol, the oracle does not compute k and sets $\pi_{\mathsf{P}}^i.\mathsf{k} \leftarrow \perp$.

Oracle $\mathsf{Test}(\pi_{\mathsf{P}}^i)$**:** If $\mathsf{P} = \mathsf{P}_*$ and $i = i_*$, parsing $\pi_{\mathsf{P}}^i.\mathsf{AID}$ as $(\Lambda'_j)_{j=1}^n$, $\mathcal{B}$ sets $\mathsf{AuthSet} \leftarrow \left\{\Lambda'_j\right\}_{j=1}^n \setminus \{\Lambda_l\}$, it computes: $\widetilde{\mathsf{k}} \leftarrow \left(\prod_{\Lambda \in \mathsf{AuthSet}} e(\hat{X}, \hat{Y}')^{\mathcal{L}'[\Lambda.\mathsf{PK}]}\right) \cdot \hat{Z}$, and returns it. Note that if $\mathcal{B}$'s challenge bit is 0, then $\hat{Z} = e(\hat{X}, \hat{Y}')^{\Lambda_l.\mathsf{SK}}$ for $\Lambda_l.\mathsf{SK}$ such that $g_1^{\Lambda_l.\mathsf{SK}} = \Lambda_l.\mathsf{pk}$, so $\widetilde{\mathsf{k}}$ is the real key expected for $\pi_{\mathsf{P}_*}^{i_*}$.

If, however, the challenge bit of $\mathcal{B}$ is 1, then $\hat{Z}$ is a random value.

Oracle $\mathsf{RevealTD}(\mathsf{sst}, \mathsf{A}, \mathsf{B}, \mathsf{O}, (\Lambda'_i)_{i=1}^n, l')$**:**

- if $\mathsf{IdentifySession}(\mathsf{sst}, \pi^{i*}_{\mathsf{P}_*}) = 1$ and $\Lambda'_{l'} = \Lambda_l$ (challenge instance, honest authority), then $\mathcal{B}$ aborts returning a random bit, as for key freshness;
- if $\mathsf{IdentifySession}(\mathsf{sst}, \pi^{i*}_{\mathsf{P}_*}) = 1$ and $\Lambda'_{l'} \neq \Lambda_l$ (challenge instance, other authority), then $\mathcal{B}$ knows the secret key of $\Lambda'_{l'}$. It acts as in G_7 except that it computes $\Lambda'_{l'}.t_1 \leftarrow e(\hat{X}, \hat{Y}')^{\mathcal{L}'[\Lambda'_{l'}.\mathsf{PK}]}$, $\Lambda'_{l'}.t_2 \leftarrow \mathsf{Sim}_{\mathsf{NIPoK}}(g_1, \Lambda'_{l'}.\mathsf{pk}, g_T, \Lambda'_{l'}.t_1)$ and $\Lambda'_{l'}.t \leftarrow (\Lambda'_{l'}.t_1, \Lambda'_{l'}.t_2)$;
- if $\mathsf{IdentifySession}(\mathsf{sst}, \pi^{i*}_{\mathsf{P}_*}) \neq 1$ and $\Lambda'_{l'} = \Lambda_l$ (non-challenge session, honest authority), then $\mathcal{B}$ parses sst as $(\omega' \| m_X \| m_Y \| \sigma^1_Y \| \sigma_X \| \sigma^2_Y \| \sigma_O)$, m_X as $X_1 \| X_2 \| \mathsf{ni}_X$ and m_Y as $Y \| \mathsf{ni}_Y$, and sets $\omega \leftarrow \mathsf{A} \| \mathsf{B} \| (\Lambda'_i)_{i=1}^n$. Note that here, $\mathsf{IdentifySession}(\mathsf{sst}, \pi^u_{\mathsf{P}_v}) \neq 1$ implies that $X_2 \| Y \neq \hat{X}' \| \hat{Y}'$, or $\mathsf{A} \| \mathsf{B} \neq \mathsf{P}_* \| \mathsf{P}_w$ (or $\mathsf{A} \| \mathsf{B} \neq \mathsf{P}_w \| \mathsf{P}_*$ depending on who plays the role of Alice and Bob), or $(\Lambda'_i)_{i=1}^n \neq \pi^{i*}_{\mathsf{P}_*}.\mathsf{AID}$. It acts as in G_7 except that:
  - if $\mathsf{A} \| \mathsf{B} \| (\Lambda'_i)_{i=1}^n \neq \mathsf{P}_* \| \mathsf{P}_w \| \pi^{i*}_{\mathsf{P}_*}.\mathsf{AID}$ (or $\mathsf{P}_w \| \mathsf{P}_* \| \pi^r_{\mathsf{P}_w}.\mathsf{AID}$, if P_* plays the role of Bob), then $\mathcal{B}$ computes $\Lambda_l.t_1 \leftarrow e(\Lambda_l.\mathsf{pk}, X)^{\mathcal{L}[(Y,\omega,\sigma_Y)]}$; $\Lambda.t_2 \leftarrow \mathsf{Sim}_{\mathsf{NIPoK}}(g_1, \Lambda_l.\mathsf{pk}, g_T, \Lambda_l.t_1)$ and $\Lambda_l.t \leftarrow (\Lambda_l.t_1, \Lambda_l.t_2)$. Here, $\mathcal{L}[(Y, \omega, \sigma_Y)]$ is always defined since $\omega \neq \mathsf{P}_* \| \mathsf{P}_w \| \pi^{i*}_{\mathsf{P}_*}.\mathsf{AID}$ (or $\mathsf{P}_w \| \mathsf{P}_* \| \pi^r_{\mathsf{P}_w}.\mathsf{AID}$ if P_* plays the role of Bob).
  - if $\mathsf{A} \| \mathsf{B} \| (\Lambda'_i)_{i=1}^n = \mathsf{P}_* \| \mathsf{P}_w \| \pi^{i*}_{\mathsf{P}_*}.\mathsf{AID}$ (or $\mathsf{P}_w \| \mathsf{P}_* \| \pi^r_{\mathsf{P}_w}.\mathsf{AID}$, if P_* plays the role of Bob) and $X_2 = \hat{X}'$, then $Y \neq \hat{Y}'$, and $\mathcal{B}$ computes $\Lambda_l.t_1 \leftarrow e(\Lambda_l.\mathsf{pk}, \hat{X}')^{\mathcal{L}[(Y,\omega,\sigma_Y)]}$; $\Lambda.t_2 \leftarrow \mathsf{Sim}_{\mathsf{NIPoK}}(g_1, \Lambda_l.\mathsf{pk}, g_T, \Lambda_l.t_1)$ and $\Lambda_l.t \leftarrow (\Lambda_l.t_1, \Lambda_l.t_2)$. In this case, $\mathcal{L}[(Y, \omega, \sigma_Y)]$ is always defined because $Y \neq \hat{Y}'$.
  - else if $\mathsf{A} \| \mathsf{B} \| (\Lambda'_i)_{i=1}^n = \mathsf{P}_* \| \mathsf{P}_w \| \pi^{i*}_{\mathsf{P}_*}.\mathsf{AID}$ (or $\mathsf{P}_w \| \mathsf{P}_* \| \pi^r_{\mathsf{P}_w}.\mathsf{AID}$, if P_* plays the role of Bob) and $Y = \hat{Y}'$, then $X_2 \neq \hat{X}'$, and $\mathcal{B}$ computes $\Lambda'_{l'}.t_1 \leftarrow e(\Lambda_l.\mathsf{pk}, \hat{Y}')^{\mathcal{L}[(X_2,\omega,\sigma_X)]}$; $\Lambda.t_2 \leftarrow \mathsf{Sim}_{\mathsf{NIPoK}}(g_1, \Lambda_l.\mathsf{pk}, g_T, \Lambda_l.t_1)$ and $\Lambda_l.t \leftarrow (\Lambda_l.t_1, \Lambda_l.t_2)$. In this case, $\mathcal{L}[(X_2, \omega, \sigma_Y)]$ is always defined because $X_2 \neq \hat{X}'$.

At the end of the game, $\mathcal{B}$ forwards the bit b_* received from $\mathcal{A}$. The game is perfectly simulated for $\mathcal{A}$, and $\mathcal{B}$ wins its BDDH challenge with the same probability as $\mathcal{A}$ wins G_7, which concludes the proof of the claim. Finally, by composing the probability of all the games, we obtain the bound of $\mathsf{Adv}^{\mathsf{KS}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$.

Theorem 2. *If our protocol is instantiated with an EUF-CMA signature scheme* DS*, then our protocol is non-frameable. Moreover, for all PPT adversaries* $\mathcal{A}$*, doing at most* q_{r} *queries to the oracle* $\mathsf{Register}$*, we have:*

$$\mathsf{Adv}^{\mathsf{NF}}_{\mathsf{LIKE},\mathcal{A}}(\lambda) \leqslant q_{\mathsf{r}} \cdot \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\lambda).$$

Proof sketch. To win the non-frameability experiment, the adversary has to build a valid session state sst for a given user, containing a valid signature of this user. We prove this theorem by reduction: assuming that an adversary is able to break the non-frameability, since this adversary generates a valid signature for a user, we can use it to break the EUF-CMA security.

Proof. **(Th2. Non-frameability).**

We use the following sequence of games.

Game G_0: The original game $\mathsf{Exp}^{\mathsf{NF}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$.

Game G_1: We denote by P_i the i-th party instantiated by $\mathsf{Register}$. Game G_1 runs as G_0 except that it begins by picking $u \xleftarrow{\$} [\![1, q_{\mathsf{r}}]\!]$. If $\mathcal{A}$ returns $(\mathsf{sst}, \mathsf{P})$ such that $\mathsf{P} \neq \mathsf{P}_u$, then G_1 aborts and returns

0. The advantage of $\mathcal{A}$ on G_1 increases w.r.t. that in G_0 by a factor equalling the probability of correctly guessing u:

$$\mathbb{P}\left[\mathcal{A} \text{ wins } \mathrm{G}_0\right] \leqslant q_{\mathsf{r}} \cdot \mathbb{P}\left[\mathcal{A} \text{ wins } \mathrm{G}_1\right].$$

We prove that $\mathbb{P}\left[\mathcal{A} \text{ wins } \mathrm{G}_1\right] \leqslant \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\lambda)$ by reduction. Assume there exists an adversary $\mathcal{A}$ that wins G_1 with probability $\epsilon_{\mathcal{A}}(\lambda)$ by returning $(\mathsf{sst}_*, \mathsf{P}_*)$ such that $\exists\ (\mathsf{A}, \mathsf{B}) \in \mathsf{USERS}^2$, $\mathsf{O} \in \mathsf{OPS}$, $n \in \mathbb{N}$ and $(\Lambda_i)_{i=1}^n \in \mathsf{AUTH}^n$ such that:

- $\mathsf{P}_* = \mathsf{P}_u$; $\mathsf{P}_* \in \{\mathsf{A}, \mathsf{B}\}$; $\mathsf{P}_*.\gamma = 0$;
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{sst}_*, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{O.PK}, (\Lambda_i.\mathsf{PK})_{i=1}^n) = 1$;
- $\forall i$, if $\pi^i_{\mathsf{P}_*} \neq \perp$ then $\mathsf{IdentifySession}(\mathsf{sst}_*, \pi^i_{\mathsf{P}_*}) = 0$ or $\pi^i_{\mathsf{P}_*}.\alpha = 0$.

We build a PPT adversary $\mathcal{B}$ breaking the $\mathsf{EUF\text{-}CMA}$ security of DS with non-negligible probability. $\mathcal{B}$ receives the verification key $\hat{\mathsf{PK}}$ from its challenger, initializes a set $\mathcal{L}_S \leftarrow \varnothing$, then it simulates G_1 to $\mathcal{A}$ as in the real game, except in the following situations:

Oracle $\mathsf{Register}(\mathsf{P}, \mathtt{role}, \mathsf{PK})$**:** If $\mathsf{P} = \mathsf{P}_*$ (as guessed before) and $\mathsf{PK} = \perp$, $\mathcal{B}$ sets $\mathsf{P}_*.\mathsf{PK} \leftarrow \hat{\mathsf{PK}}$.

Oracle $\mathsf{Send}(\pi^i_{\mathsf{P}}, m)$**:** If $\mathsf{P} = \mathsf{P}_*$, $\mathcal{B}$ simulates the oracle faithfully except it uses its $\mathsf{Sig}(\cdot)$ oracle to produce signatures. Depending on the role of P_* and the protocol step, it runs either $\sigma^1_Y \leftarrow \mathsf{Sig}(\omega \| m_X \| m_Y)$, $\sigma_X \leftarrow \mathsf{Sig}(\omega \| m_X \| m_Y \| \sigma^1_Y)$ or $\sigma^2_Y \leftarrow \mathsf{Sig}(\omega \| m_X \| m_Y \| \sigma^1_Y \| \sigma_X)$, where $\omega = \mathsf{A} \| \mathsf{B} \| (\Lambda_i)_{i=1}^n$, $m_X = X_1 \| X_2 \| \mathsf{ni}_X$ and $m_Y = Y \| \mathsf{ni}_Y$. The message/signature pairs are stored in $\mathcal{L}_S$.

Since $\mathsf{sid} = X_2 \| Y$, then X_2, Y, $\mathsf{A} = \mathsf{P}_*$, $\mathsf{B} = \pi^i_{\mathsf{P}_*}.\mathsf{PID}$ (or $\mathsf{A} = \pi^i_{\mathsf{P}_*}.\mathsf{PID}$, $\mathsf{B} = \mathsf{P}_*$, depending on who plays the role of Alice and Bob), and $(\Lambda_i)_{i=1}^n = \pi^i_{\mathsf{P}_*}.\mathsf{AID}$ are parts of the messages signed in σ_X, σ^1_Y, and σ^2_Y.

Oracle $\mathsf{Corrupt}(\mathsf{P})$**:** If $\mathsf{P} = \mathsf{P}_*$, then $\mathcal{B}$ aborts.

We parse sst_* as $\omega' \| m_X \| m_Y \| \sigma^1_Y \| \sigma_X \| \sigma^2_Y \| \sigma_O$, m_X as $X_1 \| X_2 \| \mathsf{ni}_X$ and m_Y as $Y \| \mathsf{ni}_Y$. For readability, we denote by M_X the value $\omega' \| X_1 \| X_2 \| \mathsf{ni}_X \| Y \| \mathsf{ni}_Y \| \sigma^1_Y$, and M_Y the value $M_X \| \sigma_X$. If $\mathsf{Verify}(\mathsf{pp}, \mathsf{sst}_*, \mathsf{A.PK}, \mathsf{B.PK}, \mathsf{O.PK}, (\Lambda_i.\mathsf{PK})_{i=1}^n) = 1$, we have that $\mathsf{SVer}(\mathsf{A.PK}, \sigma_X, M_X) = 1$ and $\mathsf{SVer}(\mathsf{B.PK}, \sigma^2_Y, M_Y) = 1$.

If $\forall i, \pi^i_{\mathsf{P}_*} \neq \perp$, then $\mathsf{IdentifySession}(\mathsf{sst}_*, \pi^i_{\mathsf{P}_*}) = 0$ or $\pi^i_{\mathsf{P}_*}.\alpha = 0$. Thus the oracle Send never outputs valid σ_X or σ^2_Y by using the $\mathsf{Sig}(\cdot)$ oracle on messages matching the session identifier $\mathsf{sid} = X_2 \| Y$, the identities A, B, and $(\Lambda_i)_{i=1}^n$ together. We have two cases. If $\mathsf{P}_* = \mathsf{A}$, then $(M_X, \sigma_X) \notin \mathcal{L}_S$. If $\mathsf{P}_* = \mathsf{B}$, then $(M_Y, \sigma^2_Y) \notin \mathcal{L}_S$. Finally, if $\mathsf{P}_* = \mathsf{A}$, then $\mathcal{B}$ returns (M_X, σ_X). If $\mathsf{P}_* = \mathsf{B}$, then $\mathcal{B}$ returns (M_Y, σ^2_Y).

The experiment is perfectly simulated for $\mathcal{A}$, and if $\mathcal{A}$ wins, then $\mathcal{B}$ returns a fresh and valid signature. Thus $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS},\mathcal{B}}(\lambda) = \epsilon_{\mathcal{A}}(\lambda)$, concluding the proof of the theorem.

Theorem 3. *If our protocol is instantiated with an EUF-CMA signature scheme* DS *and with extractable and zero-knowledge proofs/signature of knowledge, then our protocol is honest-operator. Moreover, for all PPT adversaries* $\mathcal{A}$ *doing at most* q_{r} *queries to the oracle* $\mathsf{Register}$*, we have:*

$$\mathsf{Adv}^{\mathsf{HO}}_{\mathsf{LIKE},\mathcal{A}}(\lambda) \leqslant q_{\mathsf{r}} \cdot \epsilon_{\mathsf{NIPoK}}(\lambda) + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\lambda).$$

Proof sketch. The first step of the HO proof is to design a key extractor, which takes in input a session state sst, brute-forces the discrete logarithm of Bob's Y, then computes the key as Bob would: $\mathsf{k} = e\left(\prod_{i=1}^n \Lambda_i.\mathsf{pk}, X_2\right)^y$. Our goal is to prove that this is the key the authorities would retrieve.

We first show (by reduction) that the adversary can only build by itself a valid sst (that may match a fake authority set) with negligible probability. Namely, if an adversary can output valid

signatures for an honest operator, then we can use it to break the EUF-CMA of the signature scheme.

Moreover, for any authority Λ and any values X_1 and Y, the proof of knowledge of a trapdoor ensures that $g_1^{\Lambda.\mathsf{SK}} = \Lambda.\mathsf{pk}$ and $\Lambda.t_1 = e(X_1, Y)^{\Lambda.\mathsf{SK}}$, which implies that $\Lambda.t_1 = e(\Lambda.\mathsf{pk}, X_2)^y$ and: $\mathsf{k}_* = \prod_{i=1}^n \Lambda_i.t_1 = e\left(\prod_{i=1}^n \Lambda_i.\mathsf{pk}, X_2\right)^y$. Thus, to win the HO experiment (and return a key such that $\mathsf{k} \neq \mathsf{k}_*$), the adversary must produce a proof on a false statement, which happens with negligible probability.

Proof. **(Th3. Honest operator).** For this proof, we first describe the (deterministic, unbounded) key-extractor algorithm $\mathsf{Extract}(\pi_{\mathsf{O}}, \mathsf{PPK})$ which, given as input an operator instance π_{O} and a set of public keys, outputs the same key k as the (honest) instances of Alice and Bob.

$\mathsf{Extract}(\pi_{\mathsf{O}}, \mathsf{PPK})$ works as follows:

- Parse $\pi_{\mathsf{O}}.\tau$ as $(m_X, m'_X, (m_Y, \sigma^1_Y), (m'_Y, \sigma^1_{Y'}), \sigma_X, \sigma'_X, \sigma^2_Y, \sigma_{\mathsf{O}})$ if O is the operator of Alice, $(m_X, m'_X, (m_Y, \sigma^1_Y), (m'_Y, \sigma^1_{Y'}), \sigma_X, \sigma'_X, \sigma^2_Y, \sigma'^2_Y, \sigma_{\mathsf{O}})$ otherwise, m_X as $(X_1 \| X_2 \| \mathsf{ni}_X)$, m_Y as $(Y \| \mathsf{ni}_Y)$, $\pi_{\mathsf{O}}.\mathsf{PID}$ as (A, B) and $\pi_{\mathsf{O}}.\mathsf{AID}$ as $(\Lambda_i)_{i=1}^n$,
- Set $y = 0$. While $g_2^y \neq Y$, do $y = y + 1$. Output y (which exists, since $Y \in \mathbb{G}_2$).
- For all $i \in [\![1, n]\!]$, if $\Lambda_i.\mathsf{PK} \notin \mathsf{PPK}$ then abort, else parse each $\Lambda_i.\mathsf{PK}$ as $(\Lambda_i.\mathsf{pk}, \Lambda_i.\mathsf{ni})$.
- Compute $\mathsf{k} \leftarrow e\left(\prod_{i=1}^n \Lambda_i.\mathsf{pk}, X_2\right)^y$ and return k.

This extractor is correct because k is computed exactly as for Bob in the real protocol.

Informally, in the HO security game, the adversary aims to output a session state sst and a series of trapdoors such that (i) Verify accepts the session state and (ii) when applied to sst and the trapdoors, Open returns a key matching the one extracted from $\mathsf{Extract}$ on the corresponding operator instance. The adversary can corrupt all users and all authorities, but not the operator. We use the following sequence of games:

Game G_0: The original game $\mathsf{Exp}^{\mathsf{HO}}_{\mathsf{LIKE},\mathcal{A}}(\lambda)$:

Game G_1: Let O_i be the i-th oracle party output by $\mathsf{Register}$. Game G_1 runs as G_0 except that it begins by picking $u \xleftarrow{\$} [\![1, q_{\mathsf{r}}]\!]$. If $\mathcal{A}$ returns $(j_*, \mathsf{sst}_*, \mathsf{A}_*, \mathsf{B}_*, \mathsf{O}_*, (\Lambda_{*,i}, \Lambda_{*,i}.t)_{i=1}^n)$ such that $\mathsf{O}_u \neq \mathsf{O}_*$, then the experiment returns 0. The advantage of $\mathcal{A}$ on G_1 will be minored by the advantage of G_0 multiplied by the probability of guessing correctly the operator O_*:

$$|\mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_0] - 1/2| \leqslant q_{\mathsf{r}} |\mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_1] - 1/2|.$$

Game G_2: Let $(j_*, \mathsf{sst}_*, \mathsf{A}_*, \mathsf{B}_*, \mathsf{O}_*, (\Lambda_{*,i}, \Lambda_{*,i}.t)_{i=1}^n)$ be the guess of the adversary. Game G_2 runs as G_1, except that if for all $k \in \mathbb{N}$, $\pi^k_{\mathsf{O}_*}.\mathsf{sid} \neq \mathsf{sid}_*$ or $\pi^k_{\mathsf{O}_*}.\mathsf{PID} \neq (\mathsf{A}_*, \mathsf{B}_*)$ or $\pi^k_{\mathsf{O}_*}.\mathsf{AID} \neq (\Lambda_{*,i})_{i=1}^n$, then G_2 aborts and returns 0. For any adversary $\mathcal{A}$,

$$|\mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_0] - \mathbb{P}[\mathcal{A} \text{ wins } \mathrm{G}_1]| \leqslant \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\lambda).$$

We prove this claim by reduction. Assume that there exists $\mathcal{A}$ winning G_1 with probability $\epsilon_{\mathcal{A}}(\lambda)$ by returning a guess $(j_*, \mathsf{sst}_*, \mathsf{A}_*, \mathsf{B}_*, \mathsf{O}_*, (\Lambda_{*,i}, \Lambda_{*,i}.t)_{i=1}^n)$ such that for all $k \in \mathbb{N}$, $\pi^k_{\mathsf{O}_*}.\mathsf{sid} \neq \mathsf{sid}_*$ or $\pi^k_{\mathsf{O}_*}.\mathsf{PID} \neq (\mathsf{A}_*, \mathsf{B}_*)$ or $\pi^k_{\mathsf{O}_*}.\mathsf{AID} \neq (\Lambda_{*,i})_{i=1}^n$. We recall that if $\mathcal{A}$ wins G_0, then $\mathsf{O}_*.\gamma = 0$ and $\mathsf{Verify}(\mathsf{pp}, \mathsf{sst}_*, \mathsf{A}_*.\mathsf{PK}, \mathsf{B}_*.\mathsf{PK}, \mathsf{O}_*.\mathsf{PK}, (\Lambda_{*,i}.\mathsf{PK})_{i=1}^n) = 1$. Moreover, according to the rules of G_1, if $\mathsf{O}_* \neq \mathsf{O}_u$ then the experiment returns 0. We build a PPT adversary $\mathcal{B}$ that breaks the $\mathsf{EUF\text{-}CMA}$ security of DS with non-negligible advantage. $\mathcal{B}$ receives the verification key $\hat{\mathsf{PK}}$, initializes an empty set $\mathcal{L}_S \leftarrow \varnothing$, and then simulates G_0 to $\mathcal{A}(\mathsf{pp})$ faithfully, except in the following situations:

Oracle $\mathsf{Register}(\mathsf{P}, \mathtt{role}, \mathsf{PK})$**:** If the u-th registration query has $\mathtt{role} \neq \mathtt{operator}$), or $\mathsf{PK} \neq \perp$

then $\mathcal{B}$ aborts (as required since G_1). Else $\mathcal{B}$ sets $\mathsf{O}_u \leftarrow \mathsf{P}$ and sets $\mathsf{O}_u.\mathsf{PK} \leftarrow \hat{\mathsf{PK}}$.
Oracle $\mathsf{Send}(\pi^i_{\mathsf{P}}, m)$**:** If $\mathsf{P} = \mathsf{O}_u$, then $\mathcal{B}$ simulates this oracle faithfully, except that it queries its $\mathsf{Sig}(\cdot)$ oracle to produce signatures, storing the message/signature pairs in $\mathcal{L}_S$. $\mathcal{B}$ sets $M_O \leftarrow \omega \| m_X \| m_Y \| \sigma^1_Y \| \sigma_X \| \sigma^2_Y$, runs $\sigma_O \leftarrow \mathsf{Sig}(M_O)$, stores (M_O, σ_O) in $\mathcal{L}_S$, and sets $\mathsf{sst} \leftarrow (M_O \| \sigma_O)$.
Oracle $\mathsf{Corrupt}(\mathsf{P})$**:** If $\mathsf{P} = \mathsf{O}_u$, then $\mathcal{B}$ aborts.

Finally, $\mathcal{A}$ returns $(j_*, \mathsf{sst}_*, \mathsf{A}_*, \mathsf{B}_*, \mathsf{O}_*, (\Lambda_{*,i}, \Lambda_{*,i}.t)_{i=1}^n)$, $\mathcal{B}$ parses sst_* as $(\omega_* \| m_{*,X} \| m_{*,Y} \| \sigma^1_{*,Y} \| \sigma_{*,X} \| \sigma^2_{*,Y} \| \sigma_{*,O})$ and returns $(m_*, \sigma_*) = (\omega_* \| m_{*,X} \| m_{*,Y} \| \sigma^1_{*,Y} \| \sigma_{*,X} \| \sigma^2_{*,Y}, \sigma_{*,O})$.

Note that G_1 is perfectly simulated for $\mathcal{A}$ by $\mathcal{B}$. If for all $k \in \mathbb{N}$, $\pi^k_{\mathsf{O}_*}.\mathsf{sid} \neq \mathsf{sid}_*$ or $\pi^k_{\mathsf{O}_*}.\mathsf{PID} \neq (\mathsf{A}_*, \mathsf{B}_*)$ or $\pi^k_{\mathsf{O}_*}.\mathsf{AID} \neq (\Lambda_{*,i})_{i=1}^n$, then sst_* was not returned by Send on an oracle instance, so $(m_*, \sigma_*) \notin \mathcal{L}_S$. Moreover, if $\mathcal{A}$ wins G_1, $\mathsf{Verify}(\mathsf{pp}, \mathsf{sst}_*, \mathsf{A}_*.\mathsf{PK}, \mathsf{B}_*.\mathsf{PK}, \mathsf{O}_*.\mathsf{PK}, (\Lambda_{*,i}.\mathsf{PK})_{i=1}^n) = 1$, implying that $\mathsf{SVer}(\mathsf{O}_*.\mathsf{PK}, \omega_* \| m_{*,X} \| m_{*,Y} \| \sigma^1_{*,Y} \| \sigma_{*,X} \| \sigma^2_{*,Y}, \sigma_{*,O}) = 1$, so σ_* is a valid signature on m_* for the key $\hat{\mathsf{PK}}$. We also have that $\mathsf{O}_* = \mathsf{O}_u$.

Finally, if $\mathcal{A}$ wins G_0 such that for all $k \in \mathbb{N}$, $\pi^k_{\mathsf{O}_*}.\mathsf{sid} \neq \mathsf{sid}_*$ or $\pi^k_{\mathsf{O}_*}.\mathsf{PID} \neq (\mathsf{A}_*, \mathsf{B}_*)$ or $\pi^k_{\mathsf{O}_*}.\mathsf{AID} \neq (\Lambda_{*,i})_{i=1}^n$, then $\mathcal{B}$ wins his game. Then, $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS},\mathcal{B}}(\lambda) = \epsilon_{\mathcal{A}}(\lambda)$, concluding the proof.

Game G_3: In this game, Ext denotes the knowledge extractor of $\mathsf{NIPoK}\left\{d : D_1 = g_1^d \wedge D_2 = g_T^d\right\}$. Let $(j_*, \mathsf{sst}_*, \mathsf{A}_*, \mathsf{B}_*, \mathsf{O}_*, (\Lambda_{*,i}, \Lambda_{*,i}.t)_{i=1}^n)$ be the guess of the adversary. Game G_3 runs as G_2 except that:

- G_3 begins by initializing an empty list $\mathcal{L}[\,] \leftarrow \varnothing$.
- Each time it runs $\mathsf{RevealTD}(\mathsf{sst}, \mathsf{A}, \mathsf{B}, \mathsf{O}, (\Lambda_i)_{i=1}^n, l) \rightarrow \Lambda_l.t$ to the adversary, G_3 sets $\mathcal{L}[\Lambda_l.t] \leftarrow \Lambda_l.\mathsf{SK}$.
- After the guess of the adversary, it sets $\omega_* \leftarrow \mathsf{A}_* \| \mathsf{B}_* \| (\Lambda_{*,i})_{i=1}^n$, it parses the session state sst_* as $(\omega'_* \| m_{*,X} \| m_{*,Y} \| \sigma^1_{*,Y} \| \sigma_{*,X} \| \sigma^2_{*,Y} \| \sigma_{*,O})$, $m_{*,X}$ as $X_{*,1} \| X_{*,2} \| \mathsf{ni}_{*,X}$ and $m_{*,Y}$ as $Y_* \| \mathsf{ni}_{*,Y}$.
- For each $i \in [\![1, n]\!]$ such that $\mathcal{L}[\Lambda_{*,i}.t] = \perp$, it parses $\Lambda_{*,i}.\mathsf{PK}$ as $(\Lambda_{*,i}.\mathsf{pk}, \Lambda_{*,i}.\mathsf{ni})$ and $\Lambda_{*,i}.t$ as $(\Lambda_{*,i}.t_1, \Lambda_{*,i}.t_2)$. If $1 \leftarrow \mathsf{NIPoKver}((g_1, \Lambda_{*,i}.\mathsf{pk}, e(X_{*,1}, Y_*), \Lambda_{*,i}.t_1), \Lambda_{*,i}.t_2)$, then it runs $\mathsf{Ext}(\lambda)$ on $\mathcal{A}$ in order to extract the witness w for the proof $\Lambda_{*,i}.t_2$ and sets $\mathcal{L}[\Lambda_{*,i}.t] \leftarrow w$. Else, G_3 aborts and returns 0. If $g_1^{\mathcal{L}[\Lambda_{*,i}.t]} \neq \Lambda_{*,i}.\mathsf{pk}$ or $e(X_{*,1}, Y_*)^{\mathcal{L}[\Lambda_{*,i}.t]} \neq \Lambda_{*,i}.t_1$, then G_3 aborts and returns 0.

The two games only differ if the extractor Ext fails on a valid proof of knowledge generated by $\mathcal{A}$, in any of the (at most) q_{r} calls to the extractor. Hence:

$$|\mathbb{P}[\mathcal{A} \text{ wins } G_1] - \mathbb{P}[\mathcal{A} \text{ wins } G_2]| \leqslant q_{\mathsf{r}} \cdot \epsilon_{\mathsf{NIPoK}}(\lambda).$$

Finally, we show that $\mathbb{P}[\mathcal{A} \text{ wins } G_2] = 0$. Assume that an adversary $\mathcal{A}$ wins G_2 with non-zero probability. We parse $\pi^{j_*}_{\mathsf{O}_*}.\tau$ as $(m_X, m'_X, (m_Y, \sigma^1_Y), (m'_Y, \sigma^1_{Y'}), \sigma_X, \sigma'_X, \sigma^2_Y)$, m_X as $(X_1 \| X_2 \| \mathsf{ni}_X)$, m_Y as $(Y \| \mathsf{ni}_Y)$, $\pi^{j_*}_{\mathsf{O}_*}.\mathsf{PID}$ as (A, B) and $\pi^{j_*}_{\mathsf{O}_*}.\mathsf{AID}$ as $(\Lambda_i)_{i=1}^n$. On the other hand, we parse sst_* as $(\omega_* \| m_{*,X} \| m_{*,Y} \| \sigma^1_{*,Y} \| \sigma_{*,X} \| \sigma^2_{*,Y} \| \sigma_{*,O})$, ω_* as $\mathsf{A}_* \| \mathsf{B}_* \| (\Lambda_{*,i})_{i=1}^n$, $m_{*,X}$ as $X_{*,1} \| X_{*,2} \| \mathsf{ni}_{*,X}$ and $m_{*,Y}$ as $Y_* \| \mathsf{ni}_{*,Y}$. According to the rules of the game G_1, $\pi^{j_*}_{\mathsf{O}_*}.\mathsf{sid} = \mathsf{sid}_*$ and $\pi^{j_*}_{\mathsf{O}_*}.\mathsf{PID} = (\mathsf{A}_*, \mathsf{B}_*)$ and $\pi^{j_*}_{\mathsf{O}_*}.\mathsf{AID} = (\Lambda_{*,i})_{i=1}^n$, which implies that we have $X_{*,1} = X_1$, $X_{*,2} = X_2$, $Y_* = Y$, and $\omega_* = \mathsf{A} \| \mathsf{B} \| (\Lambda_i)_{i=1}^n$. By definition, $\mathsf{Extract}$ returns $\mathsf{k} = e\left(\prod_{i=1}^n \Lambda_i.\mathsf{pk}, X_2\right)^y$, where $Y = g_2^y$. We recall that $e(X_1, g_2) = e(g_1, X_2)$.

On the other hand, the algorithm $\mathsf{Open}(\mathsf{pp}, \mathsf{sst}_*, (\Lambda_{*,i}.t)_{i=1}^n, (\Lambda_{*,i}.\mathsf{PK})_{i=1}^n)$ returns:

$$\begin{aligned}\mathsf{k}_* &= \textstyle\prod_{i=1}^n (\Lambda_{*,i}.t_1) = \prod_{i=1}^n e(X_{*,1}, Y_*)^{\mathcal{L}[\Lambda_{*,i}.t]} = \prod_{i=1}^n e(X_1, Y)^{\mathcal{L}[\Lambda_{*,i}.t]} = \prod_{i=1}^n e(X_1, g_2^y)^{\mathcal{L}[\Lambda_{*,i}.t]} \\ &= \textstyle\prod_{i=1}^n e(X_1, g_2)^{y \cdot \mathcal{L}[\Lambda_{*,i}.t]} = \prod_{i=1}^n e(g_1, X_2)^{y \cdot \mathcal{L}[\Lambda_{*,i}.t]} = \prod_{i=1}^n e(g_1^{\mathcal{L}[\Lambda_{*,i}.t]}, X_2)^y = \\ &= \textstyle\prod_{i=1}^n e(\Lambda_{*,i}.\mathsf{pk}, X_2)^y = \prod_{i=1}^n e(\Lambda_i.\mathsf{pk}, X_2)^y = e\left(\prod_{i=1}^n \Lambda_i.\mathsf{pk}, X_2\right)^y = \mathsf{k},\end{aligned}$$

which implies that $\mathsf{k}_* = \mathsf{k}$ with probability 1, so $\mathcal{A}$ cannot win the game. This concludes the proof.

7 Using our protocol

In this section we explore how our generic setup can be tailored to a number of different Lawful Interception requirements and scenarios. This highly depends on the roles of the entities involved: the operator, the authorities, and the entity that establishes a warrant, which we call a judge for simplicity. This is a gross simplification of the legal process, however, at the level of abstraction that we consider the matter, we only need that the entity's key be associated with that specific protocol instance.

The role of the authorities Lawful interception may involve multiple authorities: court of law, law-enforcement agencies, and/or other administrative agencies (including city halls, government, etc.). For this scenario, we assume the operator is *not* an authority. We call an authority *privileged* if it is allowed to view the output of the opening algorithm. We explicitly assume that authorities behave honestly in the dissemination of their trapdoors, specifically: (i) no authorities disseminates its trapdoor to a non-authority; (ii) no privileged authority disseminates its trapdoor to a non-privileged authority.

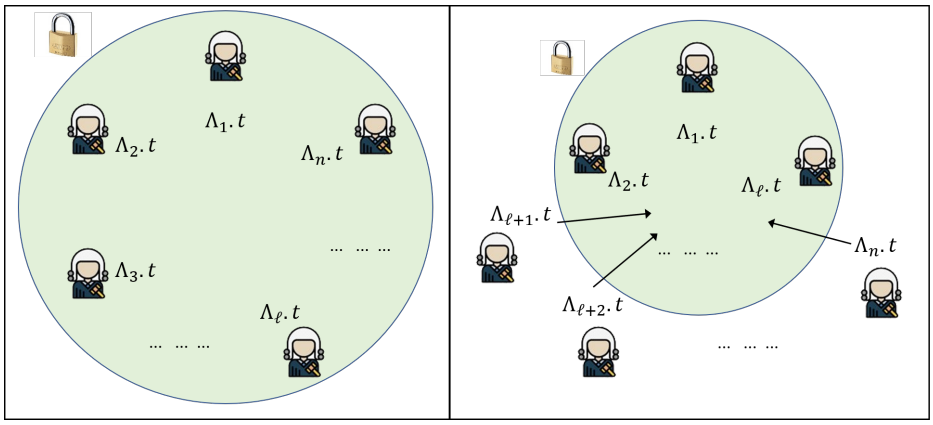

Fig. 3. An overview of the opening procedure for the case of n privileged authorities (left) and that of ℓ out of n privileged authorities (right). The circle and lock denote a secure channel shared by the parties within the circle.

Since privileged authorities are the only entities that must see sensitive data, such as the reconstructed key and the communication contents it unlocks, they are more computationally involved in key recovery. Take the case of n out of n privileged entities. To protect user privacy, they must first exchange their trapdoors over a secure channel (otherwise, an adversary may gain the trapdoors and open the key itself). Then, each authority must perform the opening procedure, involving the verification of the soundness of the trapdoors and possibly the session state.

Unprivileged authorities have a much lighter computational burden, while also being essential to the LI process. As long as there is at least one privileged authority, the only thing unprivileged authorities must do is broadcast their trapdoors. In the case of 1 out of n privileged authorities, the privileged party receives all the trapdoors, then computes its own, and finally opens the key within a secure environment. For ℓ out of n privileged parties with $1 < \ell < n$, the privileged authorities receive the trapdoors of unprivileged parties, then exchange their own privileged trapdoors over

a secure channel. Finally, each privileged party runs the opening algorithm. This is also visible in Figure 3.

The role of the operator In some cases, the operator could also play the role of an authority. Apart from participating in the authenticated key-exchange, it will need to generate a trapdoor and potentially take part in the opening process. A naïve solution would be to have the operator use its operator credentials when it plays its part as an authority; however, they may have a different format, and we opt for having the operator using two distinct sets of credentials for its roles, as depicted in Figure 4. For the operator, the following steps are modified:

- **Key generation:** The operator runs $\mathsf{AKeyGen}$ in addition to $\mathsf{OKeyGen}$ to get its credentials.
- **AKE:** The operator uses its operator credentials only in the course of the AKE protocol sessions between any two users.
- **Trapdoor generation:** Unlike in our original case, the operator will now also need to generate trapdoors for lawful interception, using only its authority credentials.
- **Opening:** Depending on whether the operator should have access or not to the output of the opening algorithm, the operator must also take part in the opening of the session key.

As the two credentials are independent, the operator's participation in the opening procedure reveals no information about its operator credentials, and vice-versa.

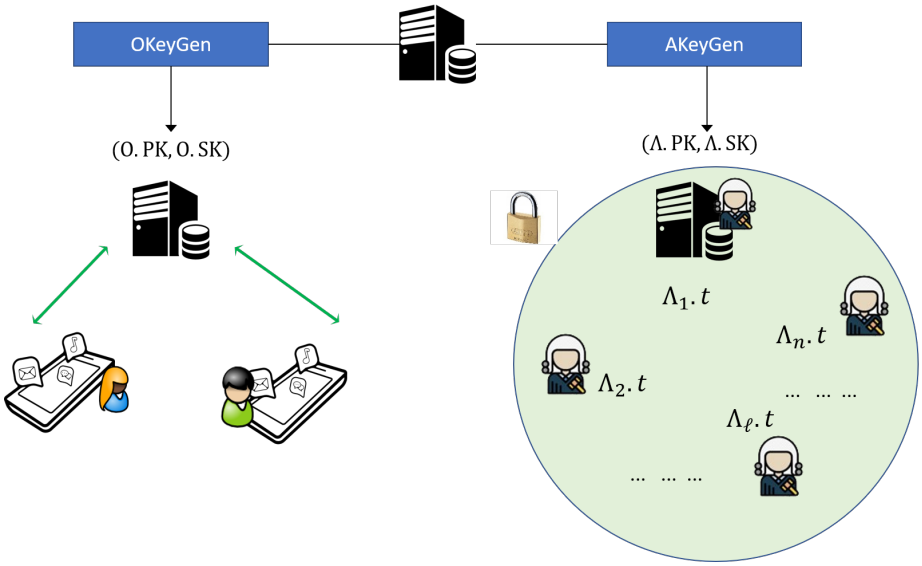

Fig. 4. When the operator is also an authority, it generates two pairs of independent credentials, using its operator credentials for AKE (left-hand side) and its authority credentials for the opening of sessions (right-hand side).

The role of the judge We can consider three types of involvement from the judge: no judge (no warrants are needed), implicit judge (provides warrant to each authority, but takes no part in the opening), and explicit judge (provides warrants and takes part in the opening). We consider a case in which the operator is not an authority, and there is a single privileged authority (*i.e.*, only one authority can learn the output of the opening algorithm). If the privileged participant is the judge, we find ourselves in the third case (explicit judge).

– **No judge:** The judge has no bearing on the setup of the protocol and may *not* be one of the authorities.
– **Explicit judge:** At the other extreme is the case of the explicit judge. Without loss of generality, we assume that the judge is Λ_1. The protocol is run as in the case of 1 privileged authority.
– **Implicit judge:** An intermediate case, in where judge must generate warrants authorizing LI, but it is not necessarily an authority. This case can be treated in two ways. The simplest solution is to consider the judge an unprivileged authority and run the protocol as indicated earlier. Then, LI cannot take place if the judge does not authorize it, but the judge does not learn the output of the opening protocol. A more complicated scenario is that in which the judge specifically does not take part in the protocol. In that case, our protocol must be composed with a different component, which provides guarantees with respect to the exact legal dependencies between the judge, its warrant, and the authorities. We consider this latter approach as possible future work.

8 Conclusion

Our paper is motivated by the observation that lawful interception does not imply that mobile users have to forgo their privacy with respect to mobile operators. To reconcile the two requirements, we introduce Lawful-Interception Key Exchange (LIKE), a new primitive that augments user privacy without harming LI. LIKE guarantees that Alice's and Bob's secure channel remains secure except with respect to: Alice, Bob, or a collusion of n legitimate authorities. Neither the operator, nor a collusion of less than n maliciously-behaving authorities can break that privacy. In addition, users are guaranteed the impossibility of being framed of wrong-doing, even if all the authorities and operators worked together. Finally, both the operator and the authorities are guaranteed that the LI procedure will reveal the key that Alice and Bob should have computed in any given session.

Our instantiated primitive relies on a digital signature scheme and NIZK proofs and signatures of knowledge for discrete logarithm. We embed the long-term credentials of the authorities on one side of a pairing, Alice and Bob's ephemeral DH elements on the other, and rely on the hardness of the BDDH problem. Contrary to existing secret-sharing approaches, in our solution the authorities do not need to store shares to Alice's and Bob's randomness for each session (which scales badly and is unreasonable given the retroactive effect of LI laws). Instead, the burden of storage rests with the operator, as is currently the case. We require no trusted party; in fact our three properties assume that the users, operators, and authorities may misbehave (in turn or concomitantly). Our protocol is versatile and its extract-and-open LI mechanism can be used in various configurations of authorities, judges, and operators.

A guarantee not provided by our protocol is against malicious users who run the protocol correctly, but then use a different key (exchanged by other means) to encrypt their communication. This is not a weakness of our scheme, but rather an inevitability as long as parallel means of exchanging keys (not subject to LI) exist.

Another limitation of our work is the fact that it only works for domestic mobile communications, where both operators are subject to the same set of authorities. It is not obvious how to extend our protocol to embed two independent sets of authority keys into the session state without risking two different conversations taking place. This extension is left as future work.

9 Acknowledgements

Cristina Onete, Olivier Blazy, and Pierre-Alain Fouque are grateful for the support of the French Agence Nationale de Recherche (ANR) through projects 16 CE39 0012 (SafeTLS) and 18 CE39 0019 (MobiS5).

References

1. 3GPP. *TS 33.126 3GPP; Technical Specification Group Services and System Aspects; Security; Lawful Interception requirements (Rel. 16)*, 09 2019.
2. 3GPP. *TS 33.127 3GPP; Technical Specification Group Services and System Aspects; Security; Lawful Interception (LI) architecture and functions (Rel. 16)*, 03 2020.
3. 3GPP. *TS 33.128 3GPP; Technical Specification Group Services and System Aspects; Security; Protocol and procedures for Lawful Interception (LI); Stage 3 (Rel. 16)*, 03 2020.
4. Stéphanie Alt, Pierre-Alain Fouque, Gilles Macario-Rat, Cristina Onete, and Benjamin Richard. A cryptographic analysis of UMTS/LTE AKA. In *Proceedings of ACNS*, volume 9696 of *LNCS*. Springer, 2015.
5. Abdullah Azfar. Implementation and performance of threshold cryptography for multiple escrow agents in voip. In *Proceedings of SPIT/IPC*, pages 143–150, 2011.
6. Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO '92*, volume 740 of *LNCS*. Springer, 1992.
7. Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. In *CCS '97*. ACM, 1997.
8. Mihir Bellare and Ronald L. Rivest. Translucent cryptography - an alternative to key escrow, and its implementation via fractional oblivious transfer. *J. Cryptology*, 12(2), 1999.
9. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO '93*, volume 773 of *LNCS*. Springer, 1993.
10. Matt Blaze. Protocol failure in the escrowed encryption standard. In *CCS '94*. ACM, 1994.
11. Xavier Boyen. The uber-assumption family. In *Pairing 2008*. Springer, 2008.
12. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO '97*, volume 1294 of *LNCS*. Springer, 1997.
13. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO*, volume 4117 of *LNCS*. Springer, 2006.
14. David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO '92*, volume 740 of *LNCS*, pages 89–105. Springer, 1992.
15. Liqun Chen, Dieter Gollmann, and Chris J. Mitchell. Key escrow in mutually mistrusting domains. In *Proceedings of Security Protocols*, pages 139–153, 1996.
16. M. Chen. Escrowable identity-based authenticated key agreement in the standard model. In *Chinese Electronics Journal*, volume 43, pages 1954–1962, 10 2015.
17. Dorothy E. Denning and Dennis K. Branstad. A taxonomy for key escrow encryption systems. *Commun. ACM*, 39(3), 1996.
18. Yvo Desmedt. Abuses in cryptography and how to fight them. In *CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*. Springer, 1988.
19. Qiang Fan, Mingjian Zhang, and Yue Zhang. Key escrow scheme with the cooperation mechanism of multiple escrow agents. 2012.
20. FCC. Communications assistance for law enforcement act. P.L. 103414, 47 U.S.C. 1001-1010, 1994.
21. Joe Kilian and Frank Thomson Leighton. Fair cryptosystems, revisited: A rigorous approach to key-escrow (extended abstract). In *CRYPTO '95*, volume 963 of *LNCS*. Springer, 1995.
22. Byung-Rae Lee, Kyung-Ah Chang, and Tai-Yun Kim. A secure and efficient key escrow protocol for mobile communications. In *Computational Science - ICCS 2001*, volume 2073 of *LNCS*. Springer, 2001.
23. Yu Long, Zhenfu Cao, and Kefei Chen. A dynamic threshold commercial key escrow scheme based on conic. *Appl. Math. Comput.*, 171(2):972–982, 2005.
24. Yu Long, Kefei Chen, and Shengli Liu. Adaptive chosen ciphertext secure threshold key escrow scheme from pairing. *Informatica, Lith. Acad. Sci.*, 17(4):519–534, 2006.
25. Keith M. Martin. Increasing efficiency of international key escrow in mutually mistrusting domains. In *Proceedings of Cryptography and Coding*, volume 1355 of *LNCS*, pages 221–232. Springer, 1997.

26. Baker McKenzie. *2017 Surveillance Law Comparison Guide*. 2017. Available at https://tmt.bakermckenzie.com/-/media/minisites/tmt/files/2017_surveillance_law.pdf.
27. Silvio Micali. Fair public-key cryptosystems. In *CRYPTO '92*, volume 740 of *LNCS*. Springer, 1992.
28. Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In *EUROCRYPT*, volume 9057, pages 657–686. Springer, 2015.
29. Crypto Museum. Clipper chip. Available at https://www.cryptomuseum.com/crypto/usa/clipper.htm.
30. Liang Ni, Gongliang Chen, and Jianhua Li. Escrowable identity-based authenticated key agreement protocol with strong security. *Comput. Math. Appl.*, 65(9):1339–1349, 2013.
31. Council of the EU. Council resolution of 17 january 1995 on the lawful interception of telecommunications. Official Journal C 329 , 04/11/1996 P. 0001 - 0006, Jan 1995.
32. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO '89*, volume 435 of *LNCS*, pages 239–252. Springer, 1989.
33. Adi Shamir. Partial key escrow: A new approach to software key escrow. Presented at Key Escrow Conference, 1995.
34. Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.
35. Zhen Wang, Zhaofeng Ma, Shoushan Luo, and Hongmin Gao. Key escrow protocol based on a tripartite authenticated key agreement and threshold cryptography. *IEEE Access*, 7:149080–149096, 2019.
36. Charles V. Wright and Mayank Varia. Crypto crumple zones: Enabling limited access without mass surveillance. In *Proceedings of EuroS&P 2018*. IEEE, 2018.
37. Adam L. Young and Moti Yung. Kleptography from standard assumptions and applications. In *Proceedings of SCN*, pages 271–290, 2010.