(P)KT-IEE: Secure Key Transparency Protocols for Interoperable End-to-End Encrypted Message Systems

Neenu Garg
School of Informatics, University of Edinburgh
United Kingdom
ngarg2@ed.ac.uk

Tariq Elahi
School of Informatics, University of Edinburgh
United Kingdom
t.elahi@ed.ac.uk

ABSTRACT

End-to-End-Encrypted (E2EE) messaging services are a key privacy enhancing technology enabling free and open speech on the Internet. They are widely deployed and very popular with large userbases. E2EE relies critically on the trustworthy distribution and storage of users' public keys. To that end, Key Transparency (KT) has been deployed by popular platforms (such as Whatsapp) and many designs and refinements have been proposed. However, KT in the **interoperable** E2EE setting has not yet been investigated. We address the challenge of distributing and trusting keys across platform boundaries and propose a Secure Key Transparency Protocol for Interoperable End-to-End Encrypted Message Systems *((P)KT-IEE)*. We also present a privacy preserving variant of our proposed protocol. This work is timely since the EU's Digital Markets Act obliges E2EE messaging platforms to allow users from different services to be able to communicate with each other. Our security and performance analysis show that our protocols are secure, private, resist local surveillance, and practical (allowing for trade-offs between light-weight and privacy preservation).

KEYWORDS

key transparency, interoperability, end-to-end encryption, key distribution, message service provider systems

1 INTRODUCTION

Platform **interoperability** between end-to-end encrypted (E2EE) messaging services will enable a user of one E2EE service provider (SP) to communicate with a user of another E2EE service provider, for example Whatsapp and Signal, without the need to create accounts on both services. The European Union's Digital Markets Act (*DMA*) came into force in November 2022, and Article 7 obligates *gatekeepers* (service providers of large message platforms) to allow interoperability of number-independent interpersonal communications services [1]. There are a number of benefits: eradicating platform monopolies, supporting smaller platforms and thus choice, and empowering users to use SPs of their own choosing and not because of social and market pressures to go where other users are. Indeed, platform interoperability will contribute to freedom of speech by enabling users to form communities online irrespective of censorship in their countries for certain messaging apps [9].

Free and Open Communications on the Internet 2024 (2), 68–76

E2EE messaging depends on the security and authenticity of users' secret and public keys. When Alice wants to communicate with Bob on the same platform, she depends on the platform to securely provide Bob's authentic public key. Both Alice and Bob have to trust that their platform is acting honestly.

Key Transparency is a mechanisms that has recently been adopted by platforms, like Whatsapp, to provide this trust to their own users. To realize interoperability in E2EE messaging platforms a "key" challenge is how to design transparent mechanisms for distributing and authenticating users' public keys across SP boundaries. Our proposed protocols, *(P)KT-IEE*, allow users of one SP to verify the authenticity of the public keys of users of another SP with trade-offs between performance and privacy.

Our Contributions. In this work, we present *KT-IEE*, a secure *KT* protocol, and its private variant *PKT-IEE*, which are, to the best of our knowledge the first for interoperable end-to-end message service provider systems. Below, we elaborate our contributions:

- A comprehensive problem description of the KT challenge in interoperable communication systems,
- The *KT-IEE* protocol design,
- An extension, *PKT-IEE*, that provides sender-recipient privacy unlinkability where the sender's SP does not learn the identity of the recipient on another SP,
- A security analysis shows resilience against potential security breaches,
- A discussion on performance of *KT-IEE* shows that there is not a significant computation overhead for clients and servers.

2 BACKGROUND AND RELATED WORK

2.1 E2EE and Key Transparency

E2EE ensures that communications between an SP's users remain private, and no one, not even the SP can eavesdrop on them. The public/private keys are the essence of E2EE messaging service. When a new user registers with an E2EE message service system, the user's device generates public/private key pairs. The public key is sent to the SP where it is stored in a secure key-store server. The private key never leaves the user's device and is stored in a secured area on the device.

When a user wishes to send a message to another user for the first time, the message is encrypted on the sender's device by first retrieving the recipient's public key from the SPs key-store server. When the message reaches the recipient's device, it is decrypted using the private key which is stored locally on recipient's device.

This encryption process establishes a secure communication channel that safeguards sensitive information. To reply, the recipient performs the same steps as above.

In a way, the secure encrypted communication in messaging services completely rests on trusting the service provider to provide the correct public keys. A compromised SP (either maliciously or through legal or coercive pressure) could replace a user's public key by one created by the service provider (i.e. the private key is known to them and can decrypt messages). This undermines trust in E2EE platforms, and as a response numerous solutions for Key Transparency have been proposed by researchers and industry. However, there has been little attention towards the trust issues regarding public key storage and distribution in interoperable E2EE messaging systems. While there are a myriad of challenges to interoperability in E2EE messaging, ranging from standardization to spam abuse and content moderation to market competition [1], we focus on providing trust for public keys through key transparency across service providers.

We first provide brief technical details of *CONIKS* [7], on which *(P)KT-IEE* is based and then we discuss closely related literature.

2.1.1 CONIKS. *CONIKS* is an end-user key verification service for sensitive communication systems that obviates the need for a global third party auditors thereby allowing the clients to efficiently monitor their key bindings for consistency. In this section we will highlight the core data structure design and key look-up operations in *CONIKS*. Interested readers may refer to the technical paper [7] for further details.

Core Data Structure. In *CONIKS*, clients' ids (usernames, phone numbers, or other unique identifier) are bound to their private keys in a key-value directory. This directory is realized as a Merkle prefix tree [8] of all registered id-key bindings. This binary tree has three different types of nodes: interior, empty, and leaf, where the contents of each are hashed values using some collision-resistant hash function H(). The leaf node is a cryptographic commitment that binds the name (id) and public key data together. The root of the tree is signed with the private signing key of the SP and this signed tree root (the *STR*) is stored at a publicly accessible location. A new *STR* is regenerated whenever there is a change to the tree (either when it is rebuilt or when an item is inserted).

Key Validation. Once *Alice* creates an account with an E2EE messaging platform (say *xyz.com*), her public key is sent to the SP where it is inserted into the KT directory (Merkle prefix tree in the case of *CONIKS*) thus registering the name-to-key binding. To validate that her key was correctly inserted into the tree, Alice requests a proof from the SP's directory. This proof is composed of the *STR* and an *authentication path* (or $auth\_path$) from the leaf node for Alice's id-key binding to the root. The $auth\_path$ is a list of hash values of the interior nodes. The particular interior nodes that are required depend on which leaf node to be verified, where each interior node provides the hash value required to calculate the value of the node higher up the tree. *Alice* thus calculates STR' and if $STR' = STR$ then she is, with high probability, sure that her key has been correctly inserted into the tree and the public *STR* is linked to it. *Alice* continues to validate her keys periodically through this process. Please see the Appendix A for details of this operation.

Key Look-up. When *Bob* wants to communicate with *Alice* (both on *xyz.com*), he performs the following steps:

- *Bob* looks up *Alice*'s public key by her username at *xyz.com* directory. He receives *Alice*'s public key along with the $auth\_path$ for that key. Using the same validation steps as *Alice* above, he verifies the proof of inclusion for *Alice*'s key using the $auth\_path$ provided and the *STR*.
- If validation is successful, *Bob* uses the provided public key for *Alice* to encrypt his message to her and then sends the message.

2.2 Related Work

The authenticity of public keys is of critical importance not only to E2EE messaging but also to e-commerce. The roll out of SSL/TLS and the certificate authorities that play a part in the distribution architecture for public key cryptography enabled merchants and customers to conduct transactions without fearing eavesdroppers stealing credit card and other sensitive information. However, the authenticity of the public keys that SSL/TLS depend on is susceptible to rogue or compromised certificate authorities, with cases of fraudulently issued certs (i.e. public keys) allowing MITM and other attacks. To mitigate that threat, Certificate Transparency Logs (*CT*) [5] is the public, auditable, append-only certificate log. It creates an untrusted log of all certificates (i.e. public key to domain name bindings) issued that is cryptographically verifiable. Anyone may check that certificates exists in the log, and web servers can monitor the log for any discrepancies for certificates issued for their own domains. The security of the log and its hosts is critical hence distributing it across several locations and jurisdictions mitigates the threat of a single point of failure. Another approach [5] is to write the CT to an append-only distributed ledger (aka blockchain) that adopts X.509 certificate registration and validation methodology for SSL/TLS. It augments the chain-of-trust model by providing support for multi-layer scrutiny of the entire existing certificate system. Furthermore, it drops seamlessly into existing SSL/TLS protocols and is much simpler than other existing public log-based approaches.

Certificate logs allow anyone to enumerate every domain with a certificate in the log. This is a privacy issue in the E2EE setting because the size, membership, and system identifiers (e.g. phone numbers or real names) of the user-base of a platform is sensitive information. *CONIKS* [7] was the first protocol that avoids those privacy issues in key transparency. Like certificate logs, key transparency logs also require secure storage, thus [4] extends *CONIKS* by adopting append-only distributed ledger technology to reach consensus on the state of the key logs built by a key server, thus reducing collusion and single-point-of-failure risks. Another blockchain-based solution [2] relies on the Ethereum [11] protocol, which is a decentralized platform in which it is possible to reach consensus also about the state of some code using the smart contract paradigm.

Chase et al. [3] propose *SEEMless*, which provides a more rigorous security analysis than the prior work and can be seen as an extension to *CONIKS*. They introduce a primitive called *verifiable key directory* (*VKD*) with definitions of the required security properties and proofs of correctness of the underlying scheme. Their

implementation relies on local *RAM* storage to store all of the data required for key transparency. Parakeet [6], aims to overcome the scalability challenges of prior academic work that limit their usefulness for real-world E2EE messaging applications. Their ordered Zero-Knowledge Set (*oZKS*) cryptographic construction allows more efficient *VKD* with storage improvements of up to an order of magnitude over prior work.

Len et al. formally analyse and produce a mapping from the DMA to an abstract API to capture functionality for interoperable E2EE. They then use this high-level API to propose a possible design for an interoperable E2EE service. Our work is thus complementary and can be seen as fitting into their high-level *service name discovery* component with KT-IEE as a concrete design incorporating KT into E2EE, thus augmenting their *retrieve key material* functionality.

Our proposed protocols, *(P)KT-IEE*, based on *CONIKS*, are a concrete first step towards enabling trustworthy cross-platform E2EE communications.

3 SYSTEM MODEL

There are four entities involved in *KT-IEE*:

(1) E2EE message service providers X (SP_X) and Y (SP_Y): Each service provider runs its own instance of a key transparency directory.
(2) Alice (C_a): *Alice* is a client of SP_X. *Alice*'s device periodically validates her key bindings at SP_X's KT directory and can alert her in case of any unexpected keys found bound to her username.
(3) Bob (C_b): *Bob* is a client of SP_Y. *Bob*'s device periodically validates his key bindings at SP_Y's KT directory and can alert him in case of any unexpected keys found bound to his username.

SP_X and SP_Y relay messages and distribute public keys among their own clients. They also distribute public keys of their clients to other service providers when requested.

3.1 Problem Statement

When C_b wants to communicate with C_a, he needs her public key for secure E2EE communications. C_b does a key look-up request for C_a's username (or some other unique identifier used by her service) at SP_X via SP_Y. C_b receives a response from SP_X via SP_Y. There is a need of a protocol, initiated by C_b's device b_d, to satisfy C_b that the response is authentic and the public key it receives actually belongs to C_a.

3.2 Assumptions

In this section, we outline the assumptions for service providers and their clients.

3.2.1 *Service Provider Assumptions.*

- Service providers have a reputation to protect and do not attack their clients in a publicly detectable manner. To provide assurances to their clients they enable KT mechanisms that allow their clients to self-verify the public keys of clients belonging to other message service provider systems by using *KT-IEE*.
- We assume the existence of a PKI whereby each service provider can distribute their public encryption and verification keys to users of their own platform and users of other platforms. One way to achieve this is to "bake-in" keys into client software.
- Service providers act as "proxies" for communications between their users and those of other platforms. In other words, clients do not directly connect to each other's devices over the network. A consequence of this is that censorship cannot be targeted against particular pairs of users by a network adversary.
- We assume that both service providers are able to be accessed by both their own users and also each other. In the case that one service provider is accessible, but the other is not (due to censorship for example), we assume that the service provider that is accessible is hosted outside the sphere of influence of the adversary such that that adversary cannot observe or block connection between the two providers.

3.2.2 *Client Assumptions.*

- Compromised client devices that report arbitrary results of the validation protocol or do not adhere to the E2EE protocol itself are out-of-scope.
- Clients validate their own keys frequently with their service provider and report any discrepancies in a public manner (i.e. so that other users do not trust the results of key look-up from that provider). This could be done over an out-of-band channel but, like other works in the literature, a solution is out of scope of this present work.
- Each client is registered with their SP and their public keys have been added (via KT) to the key directory and their devices hold their private key.

4 THREAT MODEL

We assume that E2EE messaging systems employ secure channels between client devices and servers operated by the service provider as well as between service providers' servers. Hence, third party interference of the secure channel is detectable and does not impact the confidentiality or integrity of the communications. We assume that the service providers may observe all messages in the proposed protocol and can create or drop messages but are unable to break the encryption and signature schemes employed to protect the confidentiality or integrity of messages. Further, they may change, drop, or create any unprotected information in the protocol messages. The adversary can also observe the contents of the key-value storage data structure but is not interested in compromising it if the key transparency functionality it enables.

The service provider desires to remain reputable but may deviate from honestly running the protocol if it is undetectable. A service provider may act maliciously due to either its own volition or due to external threats like coercion by law enforcement or repressive regimes. However, client devices, and the messaging app itself, are considered secure and honest.

4.1 Adversary Goals

The aim of the adversary is to convince a client of another service provider that a key the adversary controls is actually the key of

some honest user on its platform and vice versa (convince its own user about a key from a user on another platform). *KT-IEE* aims to prevent this situation, assuming the KT functions of each service provider are operating correctly.

If the adversary is successful in achieving its goal of convincing a user that a fake key is authentic then they may use that ability to eavesdrop on conversations (Meddler-in-the-middle, MITM) or fabricate messages (or whole conversations) with honest users (Impersonation).

We next describe *KT-IEE* and discuss how it prevents attacks in Section 6.

5 THE *KT-IEE* PROTOCOL

The objective of *KT-IEE* is to provide a secure, practical and consistent public key transparency mechanism to the users in an interoperable E2EE messaging scenario. In Table 1, we describe the notation used in the description of our proposed protocol. The schematic of all phases of *KT-IEE*: setup, key look-up in cross platform, response generation and response verification phases is depicted in Figure 3. The proposed protocol is defined in four phases described below.

Table 1: Notations and Descriptions

Notation	Description
SP_X	Service Provider X
SP_Y	Service Provider Y
C_a	*Alice*, client of SP_X
C_b	*Bob*, client of SP_Y
a_d, a_{id}	$Alice's$ device and unique identity
a, g^a	$Alice's$ private and public keys
b_d, b_{id}	$Bob's$ device and unique identity
b, g^b	$Bob's$ private and public keys
(PK_X, SK_X)	Encryption and decryption keys of SP_X
$(\sigma, \lambda = g^\sigma)$	Signing and verification keys of SP_X
(PK_Y, SK_Y)	Encryption and decryption keys of SP_Y
$(\alpha, \beta = g^\alpha)$	Signing and verification keys of SP_Y

5.1 Setup Phase

5.1.1 Client. The process starts when the messaging apps are installed by clients on their devices. Registration of C_a and C_b with SP_X and SP_Y is shown in Figure 1 and details for C_a are given in this section. Similar steps are taken by C_b's device while registration and is not shown here. The Setup phase is described below.

- Let $\mathcal{G}$ be a prime order (q) group with g being generator of the group. Let Let $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ be a secure cryptographic hash function. Here $\mathbb{Z}_q^*$ is a multiplicative group of integers modulo q.
- Let a be an arbitrary variable such that $a \in \mathbb{Z}_q^*$.
- a_d generates a public key g^a.
- a_d communicates g^a to her service provider SP_X.
- a_d stores secret key a in its secure local storage.

5.1.2 Service Provider. SP_X generates a Merkle prefix tree with all the name to-key bindings [10] of its registered clients. SP_X creates a Schnorr signature (s, R) for the root of the public key directory tree that we'll refer to as the Signed Tree Root (STR). Details follow: SP_X selects a random value k and generates a signature value R:

$$R = g^k$$

The value of s is:

$$s = k - (H(STR||R).\sigma) = k - E.\sigma$$

Here, $E = H(STR||R)$ and $||$ denotes the concatenation and H denotes a secure cryptographic hash function. Thus, the signature on STR is (s, R). SP_X publishes $(STR, (s, R))$ to a publicly accessible location. The generation of new STRs by service providers happens at regular intervals (whenever the tree is updated with new and updated keys).

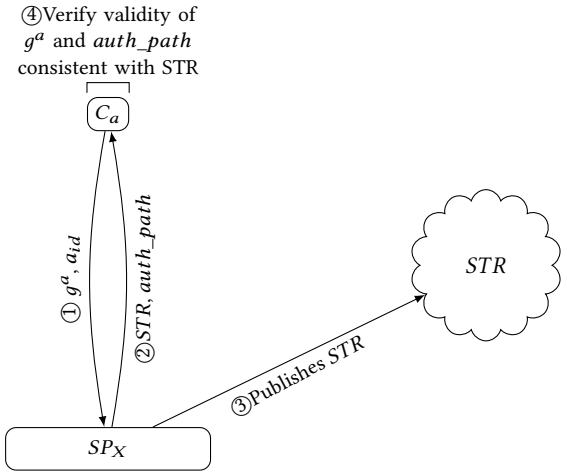


Figure 1: Steps taken when *Alice* registers with SP_X

5.1.3 Client-side Inclusion Validation. In response to a validation request, SP_X returns the full authentication path ($auth\_path$) for C_a's binding in the Merkle prefix tree. The $auth\_path$ is a proof of inclusion which reveals that an index exists in the directory corresponding to C_a. Recalling [7], the leaf nodes are not plaintext public keys, they are a cryptographic commitment to the name and key data. To prove inclusion of the full name-to-key binding, SP_X provides the opening of the commitment in addition to the authentication path. In the final step, C_a recomputes the root of the tree using the $auth\_path$ and checks that the computed root is consistent with the public STR. C_a should periodically perform this auditing to ensure the consistency of her public keys.

5.2 Cross Platform Key Look-up

Figure 2 depicts the steps taken when C_b wants to communicate with C_a. C_b sends a key look-up query, $Q(a_{id}, b_{id})$, to SP_X via SP_Y (step 1 and 2 in the figure).

5.2.1 Response Generation. In order to generate the response, SP_X first:

- Computes a string $Z \leftarrow a_{id}||g^a||auth\_path$.
- Then using Schnorr signatures, SP_X signs Z with its signing key to make sure that the recipient can verify the authenticity

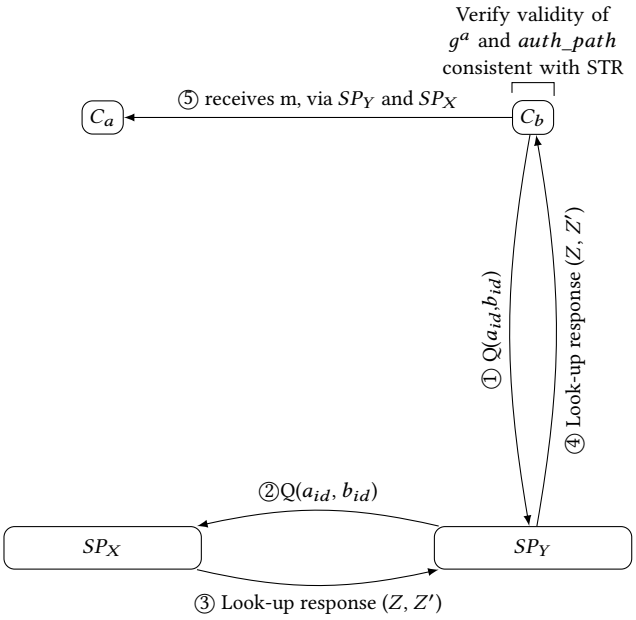


Figure 2: Steps taken when *Bob* looks up for public key of *Alice*

of response received. To sign Z, let k be any random value $\in \mathbb{Z}_q^*$, compute:

$$R = g^k$$

Compute $E = Hash(Z||R)$. The value of s is:

$$s = k - \sigma.E$$

Thus, the signature $Z' = (s, E)$.

- SP_X sends the response (Z, Z') to SP_Y which forwards it to C_b (steps 3 and 4).

5.3 Response Verification

C_b receives (Z, Z') from SP_Y and first verifies the signature Z'. Let $R_v = g^s\lambda^E$ and $E_v = Hash(R_v||Z)$. If $E_v = E$, then the signature is verified. In detail,

$$R_v = g^s\lambda^E = g^{k-\sigma.E}g^{\sigma.E} = g^k = R$$

Hence, $E_v = Hash(R_v||Z) = Hash(R||Z) = E$. After the successful verification of Z', from the string Z, b_d recovers a_{id}, g^a and $auth\_path$ and C_b confirms a_{id} was the one sent in *key look-up query* message. b_d regenerates STR using the information in the $auth\_path$. It then proceeds to verify the consistency of publicly available signed STR with the generated root tree as follows:

- Let $R_v = g^s\lambda^E$
- Let $E_v = Hash(R_v||STR)$

If $E_v = E$, then the signature is verified. If the signature is verified, *Bob* is certain about the consistency of public key of *Alice* because STR represents that a service provider is maintaining a linear history of public keys of *Alice*. *Bob* can now encrypt its message using *Alice's* public key and can then send the message. If the signature is not verified, it is possible that the SP_X has generated a fork in *Alice's STR* history or there has been a change in *Alice's* public key due to device loss or any other reasons.

5.4 E2EE Message exchange

Once the public key g^a is successfully verified by C_b, he can send the encrypted message to C_a. Let m be the message that C_b wishes to send to C_a (via SP_Y and SP_X; step 5). C_b creates:

$$M = ((Enc_{PK_X}(Enc_{g^a}(m, b_{id})), a_{id}), SP_X)$$

and sends it to SP_Y who, seeing that it is destined for SP_X, forwards just the encrypted part of the tuple. SP_X decrypts the message and sees that the inner encrypted data is for C_a and sends her $Enc_{g^a}(m, b_{id})$. C_a can then, using her private decryption key a, recover m and also see the message is from b_{id}. To reply, she can now execute key look-up using *KT-IEE* to recover the public key of C_b from SP_Y using b_{id}.

6 SECURITY ANALYSIS

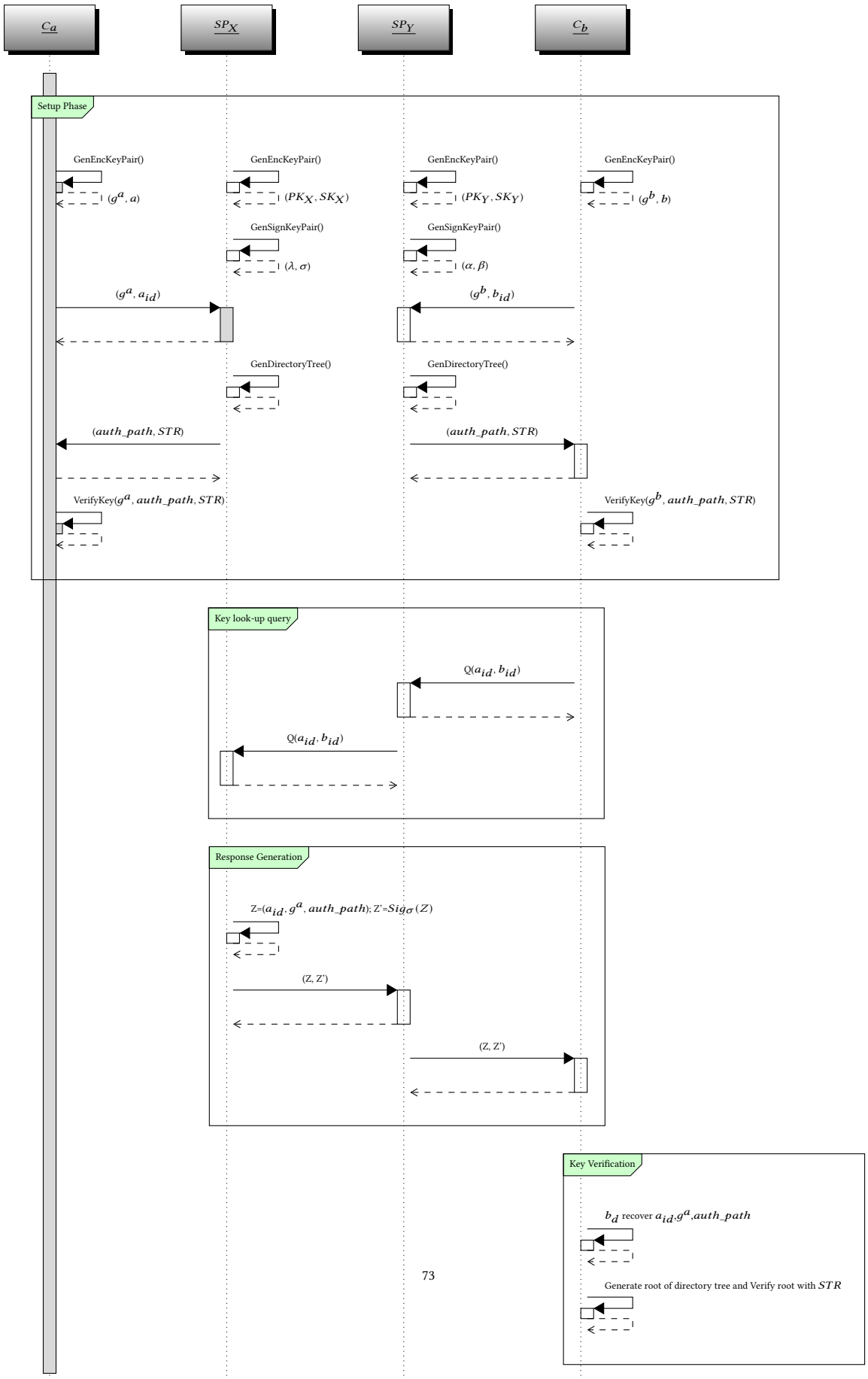
We now discuss the security of *KT-IEE* and its resistance to an adversary that wishes to convince an honest user (e.g. *Bob*) that a key the adversary controls is actually that of another honest user (e.g. *Alice*).

Let us first consider the case that *Bob*'s service provider, SP_Y, attempts this attack. Let's assume that the adversary has generated a private/public key pair y, g^y. Then, when the key look-up response, (Z, Z'), is received from SP_X, the adversary may try to replace g^a with g^y. If *Bob* accepts this change and uses g^y to encrypt messages meant for *Alice*, then SP_Y can read these messages and can forward them encrypted with the correct key g^a. This is the meddler-in-the-middle (MITM) attack and SP_Y may (selectively) use it to compromise *Bob*'s communications. However, *Bob* will not accept this change because SP_Y will not be able to produce a valid signature (Z') since it does not have access to SP_X's private signing key σ to sign the changed original response Z.

Let's now assume that SP_Y creates an account on SP_X's platform with the identifier y_{id} and key pair y, g^y, and that the binding y_{id}, g^y has been registered in SP_X's KT directory. Then in the look-up request issued by *Bob*, SP_Y can replace a_{id} with y_{id}. SP_X will respond with $(Z = y_{id}||g^y||auth\_path, Z')$. *Bob* will not accept this response since b_d checks if the identifier in the request matches the one in the response. Here they do not since $a_{id} \neq y_{id}$. SP_Y can also not replace y_{id} with a_{id} because once again it cannot then produce a valid Z' on the changed Z since it does not have access to SP_X's private signing key. Hence, it is not possible for SP_Y to conduct a MITM attack on its own users when they communicate with users from another service.

Now, let's consider that SP_X wants to MITM communications initiated by *Bob*. We will note here that unlike before, signatures do not provide any protection from SP_X since it has access to the signing key σ. Instead, we will depend on the security properties of SP_X's KT scheme.

Let's assume that SP_X generates a private/public key pair x, g^x and inserts it into its KT under the identity x_{id}—it cannot register it under a_{id} without invalidating its KT security since *Alice* is frequently checking her key bindings. Then, instead of responding


C_a
SP_X
SP_Y
C_b
Setup Phase
GenEncKeyPair()
(g^a, a)
GenEncKeyPair()
(PK_X, SK_X)
GenEncKeyPair()
(PK_Y, SK_Y)
GenEncKeyPair()
(g^b, b)
GenSignKeyPair()
(λ, σ)
GenSignKeyPair()
(α, β)
(g^a, a_{id})
(g^b, b_{id})
GenDirectoryTree()
GenDirectoryTree()
$(auth\_path, STR)$
$(auth\_path, STR)$
VerifyKey(g^a, $auth\_path$, STR)
VerifyKey(g^b, $auth\_path$, STR)
Key look-up query
$Q(a_{id}, b_{id})$
$Q(a_{id}, b_{id})$
Response Generation
$Z=(a_{id}, g^a, auth\_path)$; $Z'=Sig_\sigma(Z)$
(Z, Z')
(Z, Z')
Key Verification
b_d recover a_{id},g^a,$auth\_path$
Generate root of directory tree and Verify root with STR

with $a_{id}||g^a||auth\_path$ it sends back $a_{id}||g^x||auth\_path'$, where $auth\_path'$ is correct for x_{id}. However, *Bob* will not accept this response since the KT verification step will not complete successfully since $(a_{id}, auth\_path')$ will cause a failure. If, instead, in the response a_{id} is replaced with x_{id} then, like before, b_d will stop here since the identifiers in the request and response do not match. In effect, SP_X cannot attack users of other services without also compromising its own KT, which we have assumed is not desirable.

In real life it may be argued that *SP*s have reputations to uphold and that collusion would likely be risky and may lead to reputation loss. However, if it is undetectable then both SP_X and SP_Y may collude to compromise *Bob*'s communications with *Alice*. This means that now SP_X's signatures are no protection from SP_Y since SP_X can share σ. This situation reduces to the case above where SP_X is malicious, and again the security of the scheme rests on the security of SP_X's KT protocol. In other words, even if both service providers collude, they cannot convince *Bob* to accept a malicious key without also compromising the integrity of SP_Y's KT directory. In other words, they cannot collude to compromise each other's userbase while at the same time maintaining an illusion of safety to satisfy their own userbases.

7 *(P)KT-IEE*: PRIVATE INTEROPERABLE KEY TRANSPARENCY

In the request and response messages between *Bob* and SP_X above it is possible for SP_Y to observe which user *Bob* wants to talk with, i.e. *Alice*. Indeed, this is a privacy issue not only for *Bob* but also since it may potentially reveal to SP_Y the user identifiers and total number of users of SP_X. KT schemes like *CONIKS* protect these pieces of information from users in their own platforms. We next describe *(P)KT-IEE*, an extension of our proposal that provides the same privacy protection in the interoperable context, at the cost of additional cryptographic operations.

Let's say that *Bob* wants to communication with a user on SP_X's platform. Our extension uses Diffie-Hellman key exchange between *Bob* and SP_X to establish a shared secret key k. *Bob* uses the one-sided authenticated version of DH using SP_X's public encryption key so that SP_Y cannot interfere and succeed in MITMing this exchange. After the successful completion of the DH exchange, *Bob*'s look-up requests are now:

$$Enc_k(Q(a_{id}, b_{id})),$$

and the SP_X's look-up responses are:

$$Enc_k(Z).$$

After decrypting these messages both SP_X and *Bob* may proceed as before. Notice that the signature, Z', is no longer needed since SP_Y cannot interfere in this response.

Note that SP_X still learns that *Bob* wishes to talk with *Alice*. In the same vein, in order to reply to *Bob*'s messages, *Alice* will request *Bob*'s key from SP_Y, which will reveal to SP_Y that *Alice* is communicating with *Bob*. Further, the timing of look-up requests may allow SPs (without colluding) to infer likely communicating partners, for example, *Alice* may look-up *Bob* very soon after *Bob*'s message to her and so SP_Y may infer that *Alice* and *Bob* are communicating. Even if *Alice* never looks up *Bob*, SP_X may collude with SP_Y and share information about look-up requests it sees.

As mitigation the look-up request from *Bob* for *Alice*'s key is changed to $Enc_k(Q(a_{id}))$. Notice from Section 5.4 that b_{id} is included in every E2EE to *Alice* so she will know whose key to look-up to use with her reply. By symmetry, *Alice*'s look-up request hides her identity from SP_Y, and both *Alice* and *Bob* can communicate with each other without revealing to their own SP which user on the other platform they are communicating with. Furthermore, the look-up requests do not reveal information about each platform's userbase. It should be noted that *(P)KT-IEE* is incentive compatible in that the SP that employs it enjoys privacy for their userbase.

However, traffic correlation is still possible since SPs can collude to share information about times messages are sent and received to infer which users are talking with each other. This is outside the threat model since E2EE messaging does not protect against this type of attack. A possible avenue would be to investigate schemes that integrate *mix networks* that provide protection against timing attacks and other meta-data attacks, however this is left as future work.

8 PERFORMANCE

We now discuss the relative cost of our schemes as compared to our reference *CONIKS* [7]. We provide an analytic look at the costs to clients and identify the additional overhead our schemes impose over the baseline in terms of number of operations.

8.0.1 Key Validation. In CONIKS [7], to monitor the consistency of its own key binding in KT directory, a client downloads the *STR* (64 bytes) and the *auth_path*. Assuming N registered clients, the *auth_path* will be $\log_2 N$ hashes long, where each hash is 32 bytes. The total is $64 + \log_2 N \times 32$ bytes. The computational costs are one signature verification for the STR, and $\log_2 N$ hash and compare operations.

8.0.2 Key Look-up and Validation. A look-up and validation of another user on the *same* service is the same as above for *KT-IEE*. However, to do a cross-platform look-up under *KT-IEE*, the client needs to download the same amount as above and an additional signature Z'. This will result in one more signature verification operation than above to verify the authenticity of response message and a constant-time user_id comparison operation—to check if the ids in the request and response match.

In *(P)KT-IEE*, the Diffie-Hellman key exchange protocol is in addition to the above, with one less signature Z' to download and verify. There is also the computational overhead of the secret key encryption and decryption operations at the server and client respectively.

An implementation of KT-IEE is on our roadmap, however, to provide expected concrete figures, we refer to the empirical results reported in the *CONIKS* [7] paper. From those results we can see that, on a 2 GHz Intel Core i7 laptop, for a directory with 10 million users, verifying the authentication path for a single key binding required on average 159μs (sampled over 1000 runs, with σ = 30). Signature verification averaged 400μs, dominating the cost of authentication path verification. Thus a *KT-IEE* implementation, will impose an additional 400μs for signature verification, or an increase of 41% than [7]. From the user's perspective, DH key exchange may be completed in less than a second and comparable hardware as

above, as evidenced by the unnoticeable delay when loading webpages. Thus, while it may be a few orders of magnitude more costly than the baseline protocol, it provides privacy as a trade-off, which the service provider can reason about.

9 DEPLOYMENT CONSIDERATIONS

Interoperability in E2EE messaging raises many technical and sociotechnical challenges and issues to resolve before it sees widespread deployment. Our proposal addresses the technical challenges of trust and privacy of key transparency in this setting.

Trust Between SPs. It is likely the case that different E2EE service providers use different implementations of a KT scheme. Furthermore, it may also be the case that the KT schemes are based on different designs. These differences may result in different properties and differing degrees of trustworthiness. It is outside the scope of this proposal to provide guidance on which schemes or implementations SPs should adopt or how they rank in terms of security or privacy features. It is up to an SP to decide whether it will interoperate with another SP given what it knows about the other provider and its trustworthiness and whether there is a benefit to using *PKT-IEE* versus *KT-IEE* in that setting. In addition, users of both may also exercise their will by deciding if they will communicate across this "peering" path. In fact, the SP's choice to use the private or non-private version of the proposed schemes may also play a part in users' decision making.

Nonetheless, both *(P)KT-IEE* may be deployed in a heterogeneous setting provided that each instance of the KT directory provide a reliable and unique public *STR* that is available to the userbases of the SPs and that each SPs KT look-up service forwards messages to its counterpart's KT look-up service.

Interoperability Obligations. While the DMA applies to a specific region of the world and the SPs that operate within it, it is possible that an SP (due to market control or other business reasons) tries to avoid interoperating with other SPs. It may cite technical reasons (like those described earlier in this section) or (unfounded) trust concerns. Furthermore, a global SP (operating in many regions, some covered by DMA) may decide to provide interoperability for only those users covered by the DMA and not to the rest of its user base. While such a partitioning of the userbase may not be "good for business", these kinds of issues do highlight the impact on user freedoms and censorship inherent to the E2EE space. The DMA, with *(P)KT-IEE* as a concrete piece, provide an impetus to further normalize interoperability to the wider public.

Client Software. To realize deployment, SPs that decide to interoperate would need to provide enabling software on their client-side devices. To use the proposed schemes, SPs would implement the *(P)KT-IEE* protocols to communicate between SPs and also the verification checking algorithms required by the other SPs KT scheme. In effect, each client will have the ability to interact with not only their own KT directory but also that of SPs that their SP has agreed to interoperate with. While it may seem like a duplication of functionality and code, absent standardization, it is modular such that SPs can decide which particular other SPs (and the required algorithms) to incorporate into their own client software.

Spam Protection. Spam is a challenging issue even within a single SP. Moderation and abuse are even more challenging. We do not propose robust solutions to these latter two problems, but point to how our two proposed schemes may provide some spam protection and the trade-offs between the two. In *KT-IEE* every look-up request presents the $user\_id$ of the requester which both SPs can observe. It is then feasible to monitor users for problematic usage, e.g. looking up many usernames, sending unusually high amounts of traffic, excessive traffic to specific targets. An SP could enforce rate controls and could in the extreme opt not to respond to look-up requests from such abusive users. Of course, an SP should exercise caution and hold a high bar to banning users. Also, a user may direct its SP to only respond requests from its known contacts (which may be known from another mechanism, such as used by E2EE apps for contact discovery).

In contrast, *PKT-IEE* makes it impossible for a service provider to know the identity of the user from the other SP from the messages, since b_{id} is no longer included or visible. This also means that restricting messages to known contacts is also not possible. In that event, the user, on their own device, could filter out such abusive users (i.e. drop messages at the device). This is the trade-off of heightened privacy and is an additional consideration for an SP to interoperate with another SP. Furthermore, a malicious user or SP may try to get specific users on another service banned, or, through spurious key discrepancy reports, aim to lower the trust of a target SP. Both of these issues depend on the robustness of the reporting process (of both spam and key discrepancy). There are mechanisms in the literature like message franking and user reporting features, some privacy-preserving, that could be incorporated, however, we do not analyze these in the context of our proposal and leave as future work.

10 CONCLUSIONS

We proposed two schemes, one light-weight (*KT-IEE*) and the other relatively heavier but still practical with privacy properties (*PKT-IEE*), to address the problem of key transparency in the interoperable E2EE messaging setting. This work is the first of its kind, as far as the authors are aware, to address this challenge. With these schemes, we make progress towards enabling E2EE messaging platforms to meet the legal requirements as defined by the Digital Markets Act to allow the users of platforms to communicate across service boundaries. Our security and performance analysis show that both of our schemes are feasible to deploy, modulo clients being updated to speak our protocols and to be able to perform verification checks on other SPs' KT directories. *PKT-IEE* ensures that cross-platform surveillance is mitigated at a cost of additional computation (while remaining practical). Indeed, cross-platform user communications are more private than same-platform communications since the SP cannot see who the recipient of messages is on the other platform, however it does not attempt to resist traffic analysis by colluding SPs. We identify resistance to traffic analysis and (private) user reporting in this setting as promising lines of research.

REFERENCES

[1] Jenny Blessing and Ross Anderson. 2023. *One Protocol to Rule Them All? On Securing Interoperable Messaging.* Number 174-192. Springer.

[2] Joseph Bonneau. 2016. EthIKS: Using Ethereum to audit a CONIKS key transparency log. In *International Conference on Financial Cryptography and Data Security*. Springer, 95–105.
[3] Melissa Chase, Apoorvaa Deshpande, Esha Ghosh, and Harjasleen Malvai. 2019. SEEMless: Secure End-to-End Encrypted Messaging with less Trust. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 1639–1656.
[4] Alessandro Gattolin, Cristina Rottondi, and Giacomo Verticale. 2018. Blast: Blockchain-assisted key transparency for device authentication. In *2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)*. IEEE, 1–6.
[5] Ben Laurie. 2014. Certificate transparency. *Commun. ACM* 57, 10 (2014), 40–46.
[6] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean Lawlor. 2023. Parakeet: Practical key transparency for end-to-end encrypted messaging. *Cryptology ePrint Archive* (2023).
[7] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. 2015. CONIKS: Bringing key transparency to end users. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 383–398.
[8] Ralph C Merkle. 1980. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*. IEEE, 122–122.
[9] Beatriz Pinheiro, Carmen Moutela, Juliana Silva, Maria Leonor Ferreira, and Manuel Au-Yong-Oliveira. 2022. Social Media in China and Portugal and "digital Bubbles" of Political Information. Springer, 562–575.
[10] Claus-Peter Schnorr. 1991. Efficient signature generation by smart cards. *Journal of cryptology* 4, 3 (1991), 161–174.
[11] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151 (2014), 1–32.

A AUTHENTICATION PATH GENERATION AND VALIDATION

To use *(P)KT-IEE* we assume that KT directories are constructed as Merkle binary trees, as in *CONIKS* [7], which additionally introduces the notion of private prefixes to avoid revealing usernames in the data structure. We provide a brief overview here, for further details and explanations please refer to the original paper.

Each node in the tree represents a unique prefix i. Each branch of the tree adds either a 0 (left node) or a 1 (right node) to the prefix of the parent node. Every node other than a leaf node is hashed as follows:

$$h = H(h_{l.0}||h_{r.1})$$

Here l is the left node and r is right node. Leaf nodes are hashed as follows:

$$h_{leaf} = H(dev_{id}||key||R_n)$$

Here, R_n is a random nonce. In order to provide a proof of inclusion of *Alice*'s key in the directory, the complete authentication path between *Alice*'s leaf node and the root is generated by SP_X. This is a pruned tree containing the prefix path to the requested index as shown in Figure 4. All green leaf nodes in the tree represents other clients in the directory tree. The nodes in red have been included in the *auth_path* sent by SP_X to *Bob* in the response to the key look-up request. In our example, *Alice*'s leaf node is:

$$h_a = H(a_{id}||g^a||R_n)$$

and the *auth_path* is:

$$auth\_path = \{h_a, h_c, h'', h'\}.$$

Since, for privacy reasons, h_a is a commitment and not $(a_{id}||g^a)$, the opening for the commitment, R_n, is also sent in the response allowing h_a to be recalculated and compared for equality with that in the *auth_path*. If that is successful, b_d, using the *auth_path*, recalculates the value of the root of the directory tree:

$$STR' = (H(H(H(h_c||h_a)||h'')||h'))$$

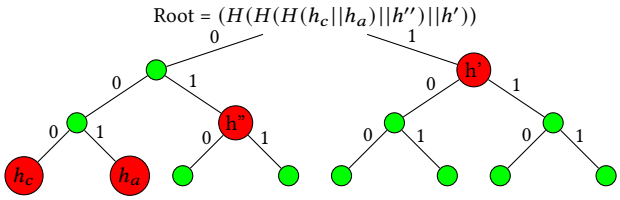


Figure 4: An authentication path from *Alice's* public key to the root node of the Merkle tree. Assuming *Alice's* index, i_{Alice} is "001"

It then performs following steps:

- Get the latest available public *STR*.
- Checks the signature on the public *STR*.
- If $STR = STR'$, then encrypt message using *Alice*'s public key, otherwise alert *Bob* of validation failure.