“Act natural!”: Having a Private Chat on a Public Blockchain

Thore Tiemann, Sebastian Berndt, Thomas Eisenbarth, Maciej Liśkiewicz

Universität zu Lübeck
{t.tiemann, s.berndt, thomas.eisenbarth}@uni-luebeck.de,
liskiewi@tcs.uni-luebeck.de

***Abstract*—Chats have become an essential means of interpersonal interaction. Yet untraceable private communication remains an elusive goal, as most messengers hide content, but not communication patterns. The knowledge of communication patterns can by itself reveal too much, as happened e. g., in the context of the Arab Spring. The subliminal channel in cryptographic systems – as introduced by Simmons in his pioneering works – enables untraceable private communication in plain sight. In this context, blockchains are a natural object for subliminal communication: accessing them is innocuous, as they rely on distributed access for verification and extension. At the same time, blockchain transactions generate hundreds of thousands transactions per day that are individually signed and placed on the blockchain. This significantly increases the availability of publicly accessible cryptographic transactions where subliminal channels can be placed. In this paper we propose a public-key subliminal channel using ECDSA signatures on blockchains and prove that our construction is undetectable in the random oracle model under a common cryptographic assumption. While our approach is applicable to any blockchain platform relying on (variants of) ECDSA signatures, we present a proof of concept of our method for the popular Bitcoin protocol and show the simplicity and practicality of our approach.**

***Index Terms*—blockchain, subliminal channel, covert communication, digital signature, information security, smart contract, steganography.**

I. Introduction

The goal of steganography is to hide information in unsuspicious documents to achieve secret communication without revealing the presence of these sensitive information. This situation is called a steganographic (or subliminal) channel – a covert channel in information processing, storage and data transmission. Modern digital steganography was first made popular due to the prisoners' problem by Simmons [1] and the investigation of steganography [2], [3], [4], [5], [6], [7] and closely related topics such as kleptography [8], [9], [10] and algorithm substitution attack (ASA) [11], [12], [13], [14], [15] have recently become the subject of intensive studies, both theoretical and empirical.

In the most basic setting, the task of the steganographic encoder (Alice) is to hide a secret message in a document, like e. g., a digital image, and to send it to the decoder (Bob) via a public channel which is completely monitored by an adversary (the warden). The goal of the encoder is that no adversary can distinguish between normal documents and documents carrying hidden information. The decoder should be able to reliably extract the hidden information from the altered documents.

Note that the goals of encryption and steganography are related, but different. Using encryption, no eavesdropper that reads the ciphertext is able to obtain the *content* of the underlying plaintext. In the steganographic setting, no eavesdroppper observing the communication between Alice and Bob should be able to detect the *presence* of the sensitive communication. One recent example for this distinction between encryption and steganography comes from the revolutionary wave in Tunisia, Libya, Egypt, Yemen, Syria and Iraq commonly known as Arab Spring [16]. Social networks and digital media played a very important role in the organization of the protests [17] and cryptographic services such as the TOR network [18] were widely utilized. While the use of these services allowed the protesters to hide the *content* of their messages, the governments of the involved countries were still able to observe communication *metadata* such as the dramatic increase of encrypted Internet traffic [19]. The reaction of some of the governments ranged from the blocking of several social media websites to the total blockade of the Internet [20]. Using steganography to hide the important protester messages in unsuspicious communication would have prevented the attention brought by the usage of cryptographic services. The importance of metadata has also been acknowledged by the NSA. General Michael Hayden, the former director of the NSA and the CIA, stressed the importance of metadata by asserting “We kill people based on metadata.” during a debate at Johns Hopkins University [21].

Another example for the importance of steganography is related to whistleblowing: If a party wants to send sensitive information to a whistleblowing platform, such as WikiLeaks, they clearly will use encryption to protect the content of these sensitive information. An observer already suspicious about the party that observes the outgoing communication of the potential whistleblower will not be able to extract the sensitive information, but the mere presence of the communication with the whistleblowing platform validates the suspicion of the attacker. If the whistleblower would embed the sensitive information steganographically in non-suspicious communication (like the upload of media to social media platforms), the observer would not be able to substantiate their suspicion.

Interestingly, in his pioneering works, Simmons uses channels in cryptographic systems as first examples for subliminal

communication, in particular in the ElGamal signature scheme [22] as well as in DSA [23]. Such subliminal channels can be applied to communicate secretly in normal looking communication using digital signatures, like e. g., blockchains.

While subliminal channels have been introduced almost four decades ago, steganography has mainly found usage in multimedia applications. Preventing subliminal communication in cryptographic systems is an important issue in cryptography research, but there have not been as many works addressing this problem [24], [25], [26], [27], [28]. On the other hand, blockchain applications have become much more prominent and nowadays generate hundreds of thousands of transactions per day which are individually signed and then stored in the blockchain. This significantly increases the availability of hidden transmissions embeddable in large publicly available data through subliminal channels in cryptographic schemes. The main focus of this paper concerns such channels in digital signatures on blockchain networks.

Recently, several works propose covert channels in digital signature schemes, particularly in ECDSA and EdDSA which are commonly used in the blockchain applications. It has been shown that both of them can be used for broadband subliminal channels [26], [29]. In [30], Ali et al. present several methods for hidden data transmission; In particular, the authors propose to reuse the randomness in ECDSA which allows the extraction of the private signing key. Frkat et al. [31] present an alternative construction, where the entire nonce can be used to transmit subliminal information to get a broadband subliminal channel. Interestingly, the algorithms proposed in both of these papers can also be used as subversion attacks, also known as ASAs, against Bitcoin (we discuss these results in more detail in Section V).

In [32], Gao et al. propose a kleptography-based digital signature algorithm to build a subliminal channel in the Bitcoin system. It uses ECDSA signatures and the OP_RETURN field to store the secret data. A drawback of this approach is that the distribution of the embedded values in that field is different from the typical distribution and might thus become detectable.

Besides ECDSA and EdDSA, several other signature schemes that could be used in blockchain applications may allow the hiding of subliminal information as well [33], [34].

Unfortunately, most of the existing subliminal channels can either be detected by analyzing their special patterns, they have low embedding rate, high time complexity, or need a previously exchanged symmetric key. Moreover, their provable security is open.

Our Contribution. In this paper we propose a public-key steganographic algorithm using ECDSA signatures on blockchain networks. We describe and implement a proof of concept of our method for Bitcoin protocols, but our approach is easily applicable for other blockchain platforms, e. g., Ethereum, Litecoin, or Dash, and for other signature algorithms like EdDSA.

In our scenario, Alice and Bob communicate subliminally by sending hidden messages in transactions, transferring bitcoins to a third non-suspicious party. They chat without having to meet a priori to agree upon a key and the third party. To this aim, additionally to the common asymmetric ECDSA signature key pairs in the wallets, both Alice and Bob hold secret and public key pairs for the hidden communication. To initiate the bidirectional channel, we propose a new way of leaking the secret signing key which is based on the following idea: In a non-interactive key exchange, using their communication key pairs, Alice and Bob share a secret which is exploited to derive a nonce during the generation of an ECDSA signature. The nonce allows to gain the secret signing key from the signature in the Bitcoin transaction. We prove that our subliminal channel is undetectable in the random oracle model under the decisional Diffie-Hellman assumption for secp256k1 and the assumption that AES is a pseudorandom permutation. This is in contrast to most previous approaches which were ad-hoc and came without any formal security model or provable security guarantee.

To the best of our knowledge, our construction is the first asymmetric stegosystem for covert communication on blockchain networks and thus prevents the need for the deployment of a high number of symmetric keys in contrast to all previous approaches. Furthermore, our approach is provably undetectable (under common cryptographic assumptions) in a formal security model similar to chosen-plaintext attacks. Finally, our stegosystem is easily implementable, very effective with constant overhead, and separates the wallet keys from the steganographic keys needed for communication, which allows a user to use multiple, independent wallets. This separation also allows for a bidirectional communication, which was explicitly out-of-scope in previous works [31].

The paper is organized as follows. Section II provides the needed preliminaries. Next, in Section III, we formally define the security model and in Section IV we provide the description and analysis of our method. In Section V we discuss in more detail the relevant, previous methods to hide messages in blockchain transactions before we conclude the paper with a short discussion.

II. Preliminaries

In this section, we give the needed preliminaries about steganographic communication, Bitcoin, ECDSA, and our cryptographic assumptions. In our pseudocode, we write $x\|y$ to describe the concatenation of two variables x and y, write $x := X$ to assign the value X to the variable x, which is final and will not change later on and write $x \leftarrow X$ to denote a non-final assignment. Finally, we write $x \leftarrow\!\!\$\, X$ for a randomized assignment, where X is a probability distribution (maybe realized by a probabilistic algorithm). We identify a finite set X with the uniform distribution on this set and thus also write $x \leftarrow\!\!\$\, X$ for such a random assignment.

A. Steganography

The goal of an encryption scheme is to hide the *content* of a message send from Alice to Bob. Using such an encryption, an observer to the communication between Alice and Bob has no way to obtain this content. Nevertheless, the observer still knows that some sensitive information between Alice and Bob were exchanged. The goal of steganography is to *embed* this sensitive information into unsuspicious communication

and thus hide the fact that Alice and Bob communicate such sensitive information.

A complexity-theoretic model for symmetric steganography was proposed by Hopper, von Ahn, and Langford [3] and, independently, by Katzenbeisser and Petitcolas [35]. The asymmetric setting was first formalized by von Ahn and Hopper [36]. The main idea behind these models is that an attacker $\mathcal{A}$ cannot distinguish between a probability distribution P, which produces unsuspicious documents, and a probability distribution Q, which embeds sensitive information into these documents. This concept of steganographic communication has found applications in covert computation, broadcasting, anonymous communication, or algorithm substitution attacks, see e. g. [37], [38], [39], [40], [41], [42], [14].

In this work, we also follow the above mentioned models. As we concentrate on developing steganographic techniques for signature schemes, we make the definitions more explicit later on. As noted above, many steganographic systems are presented without a provable security guarantee. As steganography is often employed in very sensitive scenarios (such as those described in the introduction), we believe that a provable guarantee to be of uttermost importance. This is similar to the situation for encryption, where the realization of such guarantees are essential and described as "the essence of modern cryptography, and was responsible for the transformation of cryptography from an art to a science" [43]. The Turing awards for Shafi Goldwasser and Silvio Micali also explicitly mention their "transformative work that laid the complexity-theoretic foundations for the science of cryptography" [44].

B. Bitcoin Transactions

Bitcoin is the first cryptocurrency based on a public decentralized blockchain protocol. Bitcoin was proposed by Satoshi Nakamoto in 2008 [45], first implemented in 2009, and has since then seen an enormous growth. Several further blockchain-based public ledgers have been proposed since the release of Bitcoin. Digital signatures are a fundamental building block for nearly all blockchains, due to their ability to guarantee the *authenticity* of transactions. A Bitcoin wallet is associated with at least one public/private key pair. The *private key* (also *secret key* or *signing key*), denoted sk, is to be kept secret. It is used to sign transactions issued from the corresponding wallet. The *verification key* (also called *public key*), denoted vk, is public knowledge and used to verify signatures. The *address*, denoted A, of a wallet is the hash of vk[1]. The address is used to receive transactions.

Suppose that Alice wants to pay Bitcoin to Charlie. Alice thus has a wallet with key pair $(\mathsf{sk}_A, \mathsf{vk}_A)$ and address A_A and Charlie uses $(\mathsf{sk}_C, \mathsf{vk}_C)$ as his wallet with address A_C. To pay Charlie, Alice needs Bitcoin that were sent to her wallet earlier and that they didn't spent yet. Alice then creates a *transaction*. The transaction has one or more inputs and one or more outputs. Each input references an output of a previous transaction sending Bitcoin to Alice. Each output of the transaction contains an address and the amount of Bitcoin to be sent to this address. For our example, Alice creates one output with Charlie's address A_C. Note that Alice may add an output to her own address A_A to send change back to herself. This is necessary if the sum of values defined in the inputs is greater than the amount Alice plans to send to Charlie plus some transaction fee. Alice then signs the transaction inputs and outputs with her secret sk_A and attaches the signature together with her public vk_A to the transaction. The signed transaction is broadcasted to the Bitcoin network to be mined into a block. Charlie can see that Alice sends Bitcoin their way as soon as Alice broadcasts the transaction to the network. However, Charlie can only be sure that the transaction is correct after it is mined into a block as this step involves a check of the transaction signature [46].

[1]Precisely, $A = \mathsf{Base56}(\mathsf{RIPEMD160}(\mathsf{SHA256}(\mathsf{vk}))$.

Security against existential forgery $\mathsf{G}_{\mathsf{SIG}}$.

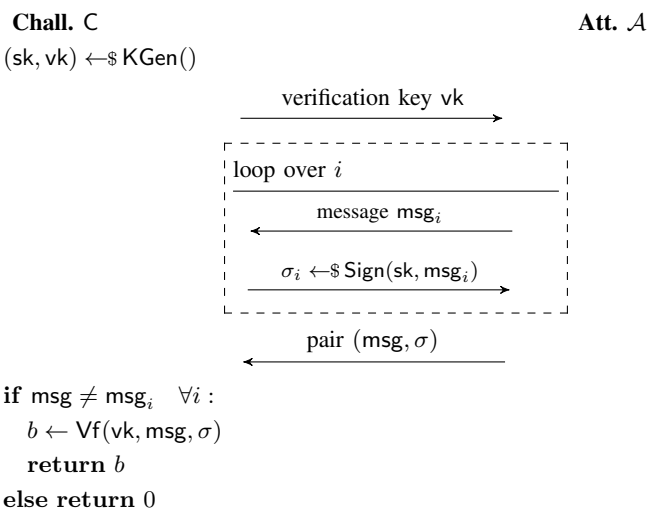

Figure 1. Cryptographic game to guarantee security against existential forgery. The attacker $\mathcal{A}$ is given a signing oracle C and their goal is to construct a valid message-signature pair by themselves.

C. Signature Schemes

In order to verify that a blockchain transaction is valid, it needs to be *signed* by the sender of the transaction. A *signature scheme* is a triple of PPTMs $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ such that Vf is deterministic and the algorithms have the following semantic:

- A call of the *key generation* algorithm KGen produces a key-pair $(\mathsf{sk}, \mathsf{vk})$ consisting of a *secret key* sk and a public *verification key* vk.
- A call of the *signing* algorithm $\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$ takes the signing key and a *message* msg (from some underlying message space) and produces a *signature* σ.
- A call of the *verification* algorithm $\mathsf{Vf}(\mathsf{vk}, \mathsf{msg}, \sigma)$ takes the verification key, a message msg, and a signature σ and outputs a bit.

We say that $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ is *correct*, if $\mathsf{Vf}(\mathsf{vk}, \mathsf{msg}, \sigma) = 1$ for all key-pairs $(\mathsf{vk}, \mathsf{sk}) \in \mathrm{Supp}(\mathsf{KGen}())$, all messages msg, and all signatures $\sigma \in \mathrm{Supp}(\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}))$.

The corresponding cryptographic game $\mathsf{G}_{\mathsf{SIG}}$ to guarantee security against *existential forgery* uses an attacker $\mathcal{A}$ as

NonceGenRFC6979(H(msg), d, cnt)

1 : $h \leftarrow \mathsf{HMAC}(d \| \mathsf{H}(\mathsf{msg}))$
2 : **for** $i = 1 \ldots cnt$:
3 : $\quad h \leftarrow \mathsf{HMAC}(h)$
4 : **return** h

SignECDSA$_{E,G,n}$(d, msg)

1 : $c \leftarrow 0$
2 : $h := \mathsf{H}(\mathsf{msg})$
3 : $k \leftarrow \mathsf{NonceGenRFC6979}(h, d, c)$
4 : $(x, y) := k \cdot G$
5 : $r \leftarrow x \bmod n$
6 : **if** $r = 0$:
7 : $\quad c \leftarrow c + 1$; goto line 3
8 : $s := [k^{-1}(h + r \cdot d)] \bmod n$
9 : **if** $s = 0$:
10 : $\quad c \leftarrow c + 1$; goto line 3
11 : **return** $(r, \min\{s, -s\})$

Figure 2. Signing algorithm as it is implemented in libsecp256k1. The message msg being signed is the transaction itself.

SignEdDSA$_{E,G,n}$(d, msg)

1 : $h := \mathsf{SHA512}(d)$
2 : $d' := 2^{254} + \sum_{i=3}^{253} 2^i h[i]$
3 : $Q := d' \cdot G$
4 : $k \leftarrow \mathsf{SHA512}(h[256:511], \mathsf{msg})$
5 : $(x, y) := k \cdot G$
6 : $r \leftarrow x \bmod n$
7 : $s := [r + \mathsf{SHA512}(r, Q, \mathsf{msg}) \cdot d'] \bmod n$
8 : **return** (r, s)

Figure 3. Signing algorithm for Ed25519 [47].

depicted in Fig. 1. During the game, the challenger C generates a key pair and publishes the public key. The attacker $\mathcal{A}$ may then request valid signatures for chosen messages. Eventually, $\mathcal{A}$ has to provide C with a signature for a message that wasn't sent to C before. $\mathcal{A}$ wins the game if the provided signature is valid. The advantage $\mathsf{Adv}^{\mathsf{sign}}_{\mathcal{A},\mathsf{SIG}}(\kappa)$ of $\mathcal{A}$ against $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ is defined as the probability that the above game $\mathsf{G}_{\mathsf{SIG}}$ outputs 1, where κ is the security parameter of the signature scheme. We say that SIG is (t, ϵ, q)-*secure*, if the term $\mathsf{Adv}^{\mathsf{sign}}_{\mathcal{A},\mathsf{SIG}}(\kappa)$ is at most ϵ for all attackers $\mathcal{A}$ running in time at most t that make q queries to the signing oracle.

D. Elliptic Curve Digital Signature Algorithm (ECDSA)

Arguably, the most widely used signature scheme in practice is the elliptic curve digital signature algorithm (ECDSA). It is also the signature scheme used to sign transactions in Bitcoin, Ethereum, and other blockchains. In order to use ECDSA, two parties agree on an elliptic curve E over a prime field $\mathbb{F}_p$ and a generator point G of order n on E. The private signing key sk, often also denoted by d, used for signing is chosen randomly from the interval $\{1, \ldots, n-1\}$ and the corresponding public key vk used for verification is chosen as the curve point $Q = dG$.

When signing a message hash h, the signer chooses a private securely random per signature nonce k from the interval $\{1, \ldots, n-1\}$ and computes the curve point $(x_r, y_r) = k \cdot G \bmod n$. The values $r = x_r$ and $s = k^{-1}(h + rd) \bmod n$ then form the signature $\sigma = (r, s)$. To verify a signature using the public key Q, the verifier computes the curve point $(x'_r, y'_r) = hs^{-1}G + rs^{-1}Q \bmod n$ and accepts the signature if $x'_r = r$ [48].

Note that all known security proofs of ECDSA rely on very strong idealized assumptions such as the generic group model (see e. g. the discussion in [49]). Nevertheless, ECDSA is the most widely used signature scheme and we will thus later assume the security of ECDSA (without explicitly stating the underlying assumptions).

1) Basic Attacks against ECDSA: Since d and k are the only two unknowns in the equation $s = k^{-1}(h + rd) \bmod n$, it is crucial to keep both values secret. If either value becomes known to an attacker, they can compute the other value as well. In particular, d is easily computed as

$$d = (s \cdot k - h) r^{-1} \bmod n \tag{1}$$

if the nonce k is revealed [50].

Also, reusing a nonce k with the same private key d and two different message hashes h_1 and h_2 immediately reveals d. Let (r_1, s_1) and (r_2, s_2) be signatures for h_1 and h_2 respectively. Since k was used for both signatures, it immediately follows that $r_1 = r_2$, because $r_1 = r_2 = (kG)_x$. This is easily detectable for an attacker. Now, the attacker can compute $k = (h_1 - h_2)(s_1 - s_2)^{-1} \bmod n$, which reveals k. With k known, they can derive the private key as above [50]

2) ECDSA in blockchains: In the context of blockchains, the elliptic curve *secp256k1* [51] is widely used when generating ECDSA signatures [52]. ECDSA signatures are malleable because whenever (r, s) verifies a hash h correctly for a given key pair, also $(r, -s)$ is a valid signature for h under the same key pair. To counter this, signatures should be normalized by only accepting signatures $(r, \min\{s, -s\})$. Bitcoin's ECDSA library *libsecp256k1*[2], which is also used by many other blockchains, performs this normalization[3].

In addition, Bitcoin recommends using a deterministic nonce generation algorithm instead of a pseudo random number generator (PRNG) to counter nonce reuse or broken PRNGs, which would allow for wallet key recovery through, e. g., lattice attacks [50] or the basic attacks described in the previous section. RFC6979 defines one way for deterministic nonce generation. As inputs it expects the hash of the message to be signed, the private signing key, and a counter which is usually chosen as zero unless the generated nonce leads to invalid values r or s during the signing process.

[2] https://github.com/bitcoin-core/secp256k1
[3] https://github.com/bitcoin-core/secp256k1/commit/0c6ab2ff

$\mathsf{ECDH}_Q(d)$

1 : $(P_x, P_y) := d \cdot Q$
2 : **if** $P_y \equiv 0 \bmod 2$:
3 : **return** $\mathsf{H}(\mathtt{0x02} \| P_x)$
4 : **return** $\mathsf{H}(\mathtt{0x03} \| P_x)$

Figure 4. ECDH key exchange in *libsecp256k1*.

Figure 2 shows the ECDSA signature generation routine SignECDSA as it is implemented in libsecp256k1. The hash function used is SHA256. ECDSA on the secp256k1 curve requires the hash to be 256 bits long, which is the case for SHA256. Therefore, we will assume that the length of the hash fits the application throughout the paper.

3) Comparison to EdDSA: In contrast to ECDSA, EdDSA chooses the nonce in a deterministic fashion by design [47]. The default curve is Curve25519 and the default hash function is SHA512. EdDSA with this configuration is called Ed25519. Many blockchains that do not rely on ECDSA use Ed25519 instead [52]. The Ed25519 signing routine is given in Fig. 3.

E. Elliptic Curve Diffie-Hellman Key Exchange (ECDH)

Let's say Alice and Bob would like to exchange a secret symmetric key k. To do so, they publicly agree on an elliptic curve E over a prime field $\mathbb{F}_p$ and a generator point G of order n on E. Alice and Bob both choose a secret random value a (resp. b) and compute $Q_A = a \cdot G \bmod n$ and $Q_B = b \cdot G \bmod n$ respectively. Next, Alice and Bob exchange their Q_A and Q_B while keeping a and b secret. Both now compute the shared point $P = (P_x, P_y)$ on E by computing $P = a \cdot Q_B = b \cdot Q_A$. The shared secret k is then chosen as $\mathsf{k} = P_x$. It is recommended to use the shared secret as an input to a key derivation function or simply a secure hash function in order to generate a shared key [53]. In the following we use the terms "shared key" and "shared secret" to describe the output of the key derivation or hash function in the context of ECDH.

1) ECDH in libsecp256k1: The library *libsecp256k1* is the reference ECDSA library for Bitcoin and has support for ECDH. We use libsecp256k1 in Section IV-D which is why we want to point out an important detail: libsecp256k1 computes k by hashing the *compressed representation* of P. This step is motivated by the potential malleability arising from the fact that both points (P_x, P_y) and $(P_x, -P_y)$ result in the same shared secret [54, sec B.4.1]. The compressed representation of P contains P_x as well as the sign[4] of P_y. So hashing the compressed representation of (P_x, P_y) results in a different hash than hashing the compressed representation of $(P_x, -P_y)$. The pseudocode of the ECDH algorithm implemented in libsecp256k1 is given in Fig. 4.

F. Cryptographic Assumptions

To show the security of our chat, we need three cryptographic assumption which are given below. We start with recalling the following definition: Two probability distributions P and Q are (t, ϵ, q)-*indistinguishable*, if every probabilistic algorithm $\mathcal{D}$ with running time t and oracle access to either P or Q that tries to distinguish them has only success probability ϵ, if $\mathcal{D}$ makes at most q queries to their oracle. We will often also say that P and Q are indistinguishable, if we are not interested in the concrete values of t, ϵ, and q.

The (t, ϵ, q)-*Decisional Diffie-Hellman assumption* (DDH) for a cyclic group $\mathbb{G}$ with generator $g \in \mathbb{G}$ and order n says that the distributions (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) are (t, ϵ, q)-indistinguishable if a, b, and c are chosen uniformly random from $\{1, \ldots, n-1\}$. A family of permutations $\{f_k\}_k$, indexed by a key k, is called a (t, ϵ, q)-*pseudorandom permutation* (PRP), if the distributions of f_k (for a randomly chosen key k) and $f^\star$ (a completely random permutation) are (t, ϵ, q)-indistinguishable. More concretely, we will assume that AES is a pseudorandom permutation. Finally, we will work in the *random oracle model* (ROM) to model the used hash function H (in our case SHA256) as a completely random oracle.

Note that the security of ECDSA (and thus of Bitcoin) already requires the random oracle model and also other assumptions which are not as standard as DDH or the hardness of AES [55], [49], [56].

Table I
NOTATION OVERVIEW. NOTE THAT ALICE HAS SEVERAL WALLETS REPRESENTED BY HER BITCOIN/ECDSA KEY PAIRS, BUT ONLY ONE CHAT KEY PAIR $(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_A^{\mathsf{Chat}})$.

Name	Meaning
$(\mathsf{sk}_i, \mathsf{vk}_i)$	Alice's i-th Bitcoin/ECDSA key pair
A_i	Alice's i-th address associated with $(\mathsf{sk}_i, \mathsf{vk}_i)$
$(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_A^{\mathsf{Chat}})$	Alice's chat key pair
$(\mathsf{sk}_C, \mathsf{vk}_C)$	Charlie's Bitcoin/ECDSA key pair
A_C	Charlie's Bitcoin address
$(\mathsf{sk}_B^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}})$	Bob's chat key pair
$\sigma = (r, s)$	An ECDSA signature
$tx_{A_i,C}$	A Bitcoin transaction from A_i to C
$\mathsf{amsg} = (\mathsf{amsg}_i)_i$	Subliminal message to be sent
$\mathsf{k}_i^{\mathsf{Chat}}$	Secret key for encrypting amsg_i
$\mathsf{ctx}_{\mathsf{Chat}}$	Encrypted amsg_i
$\mathsf{data}_{\mathsf{Chat}}$	Tuple: $(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}, \mathsf{amsg}_i, \mathsf{vk}_i)$

III. Our Model

Remember that in our setting, the goal is to transfer a message amsg from Alice to Bob. To protect against the monitoring attack described above, we assume that Alice and Bob do not use a symmetric key $\mathsf{k}^{\mathsf{Chat}}$, but work in a public-key setting. Hence, Alice has a public key $\mathsf{pk}_A^{\mathsf{Chat}}$ and a secret key $\mathsf{sk}_A^{\mathsf{Chat}}$ as well as Bob has a pair $\mathsf{pk}_B^{\mathsf{Chat}}, \mathsf{sk}_B^{\mathsf{Chat}}$ consisting of a public and a secret key. Furthermore, Alice controls several bitcoin accounts with addresses A_i and associated keys $(\mathsf{sk}_i, \mathsf{vk}_i)$ for the signature scheme / wallet. We do *not* assume that Bob has a bitcoin address himself. Hence, there is a third party, Charlie, with address A_C and keys $(\mathsf{sk}_C, \mathsf{pk}_C)$ that will receive the transactions sent by Alice. See Table I for an overview on the notation. In a concrete application, Charlie might be a party which obtains many transactions, such as a charity. To start the communication, Alice and Bob exchange their public chat keys. This can be done via a Public Key

[4]When talking about signs, we think about numbers being in $[-\frac{p}{2}, \frac{p}{2}]$. Since in practice all numbers are in $[0, p-1]$ we use "even/odd" instead of "positive/negative" to distinguish P_y and $-P_y$.

Infrastructure, during a personal meeting, or via any other out-of-band communication channel that ensures integrity and authenticity. Optionally, Alice and Bob may also agree on the Bitcoin address A_C in advance to reduce the computational costs for receiving messages.

The attacker (or warden) $\mathcal{A}$ aims to detect the presence of this hidden communication of amsg (not necessarily the content). In order to do so, they are part of the bitcoin network, i. e. they see all transactions send over the bitcoin network and can also send transactions themselves. This attacker has access to all of public information, including the public addresses A_i and the verification keys vk_i of the accounts that Alice might use to send messages, the public address A_C of Charlie, and the corresponding verification key vk_C. Furthermore, they also know the public keys $\mathsf{pk}_A^{\mathsf{Chat}}$ and $\mathsf{pk}_B^{\mathsf{Chat}}$ of Alice and Bob. To represent the fact that $\mathcal{A}$ might have previous information about the messages that Alice wants to send, we allow $\mathcal{A}$ to choose the embedded message amsg, similar to a chosen-plaintext-attack. We give a formal security game involving this attacker below.

In the following, we assume that amsg is split into parts amsg_i and each part amsg_i is sent from a wallet with key-pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ for $i = 1, \ldots, \ell$. Furthermore, we focus on the case that amsg_i can be embedded into two signatures.

A. Chat *Scheme*

Let $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be an honest implementation of a signature scheme and $\mathsf{Chat} = (\mathsf{KGenChat}, \mathsf{SignChat}, \mathsf{ExtChat})$ be a triple consisting of three PPTMs:

- The key generation algorithm $\mathsf{KGenChat}$ is used to produce the key-pairs $(\mathsf{pk}_A^{\mathsf{Chat}}, \mathsf{sk}_A^{\mathsf{Chat}})$ and $(\mathsf{pk}_B^{\mathsf{Chat}}, \mathsf{sk}_B^{\mathsf{Chat}})$.
- The modified signing algorithm $\mathsf{SignChat}$ is given the secret signing key sk, a message msg, the public key $\mathsf{pk}_B^{\mathsf{Chat}}$ of Bob, the secret key $\mathsf{sk}_A^{\mathsf{Chat}}$ of Alice, and the hidden message amsg_i and produces a signature σ for msg.
- The extraction algorithm $\mathsf{ExtChat}$ is used by Bob to extract a message amsg' from several signatures σ_i. To do so, it is given the public verification key vk, the public key $\mathsf{pk}_A^{\mathsf{Chat}}$ of Alice, and the secret key $\mathsf{sk}_B^{\mathsf{Chat}}$ of Bob.

We always assume that Chat is correct, i. e. we require that $\mathsf{ExtChat}$ is able to reconstruct the original message amsg.

Note that a common recommendation to achieve pseudonimity is to use a Bitcoin address for as few transactions as possible. Hence, sending many messages from the same Bitcoin address is detectable and thus suspicious behavior.

B. Security Model

To guarantee security, an attacker against our embedding strategy should not be able to distinguish between ℓ independent users who each send q transactions from their respective addresses A_i from Alice controlling the users and sending signatures containing steganographic material. For the sake of simplicity of the presentation, we will fix q to be two. The attacker will also be able to choose the messages msg for which the signatures should be created and the messages amsg embedded into them. Note that in Bitcoin, each transaction is unique. To model this fact, we require that the messages msg queried by the attacker to be distinct.

Formally, this distinguishability game can be described as shown in Fig. 5. Here, the challenger C generates two message key pairs and ℓ signature key pairs, sends all public keys to the attacker $\mathcal{A}$, and generates a random bit. For each signing key, $\mathcal{A}$ now provides C with two cover messages and one embeddable message and C signs both cover messages. Depending on the random bit, C optionally embeds the embeddable message into both cover messages during signing. Either way, the two signatures are returned to $\mathcal{A}$ who now has to say whether the signatures contain the embeddable message or not. Note that each signature key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ belongs to her own wallet, but all of the wallets are controlled by Alice, who has the key pair $(\mathsf{pk}_A^{\mathsf{Chat}}, \mathsf{sk}_A^{\mathsf{Chat}})$. We denote the i-th address (associated with $(\mathsf{sk}_i, \mathsf{vk}_i)$) by A_i.

We say that a system Chat is (t, ϵ)-*undetectable for* ℓ *messages* (with respect to SIG) if the probability that $\mathsf{G}_{\ell,\mathsf{SIG},\mathsf{Chat}}$ outputs 1 is at most $1/2 + \epsilon$ for all attackers $\mathcal{A}$ with running time t. Note that if Sign and $\mathsf{SignChat}$ are indistinguishable, this is sufficient to argue for security in our model.

In the context of Bitcoin, we also want to prevent against a malicious Bob who wants to steal the bitcoins of Alice. They therefore know $\mathsf{pk}_B^{\mathsf{Chat}}$, $\mathsf{sk}_B^{\mathsf{Chat}}$, $(\mathsf{vk}_i)_i$, and $\mathsf{pk}_A^{\mathsf{Chat}}$, but neither $\mathsf{sk}_A^{\mathsf{Chat}}$ nor any sk_i. As Bob knows that Alice wants to communicate with them, there is no need to hide this fact. Their goal is rather to extract bitcoins owned by Alice. A common way to leak information via ECDSA signatures also used in many of the previous works is to reveal the signing key sk_i and use it to recover the nonces k used in previous transactions (which then include information about amsg). A side effect of this is that it allows Bob to impersonate Alice by signing messages in their name, which, in the context of Bitcoin, allows Bob to obtain all of the bitcoins of Alice. To protect against such a malicious Bob, we thus need to guarantee that Alice has spent all of their bitcoins before sk_i is revealed to Bob.

IV. Subliminal Chat

We assume the following scenario: Alice sends a hidden message amsg to Bob by transferring bitcoins to Charlie. Alice has ℓ wallets, where wallet i has key pairs $(\mathsf{sk}_i, \mathsf{vk}_i)$ and address A_i. Additionally, Alice holds a key pair $(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_A^{\mathsf{Chat}})$ which we will call the "chat key pair". Charlie has a wallet with key pair $(\mathsf{sk}_C, \mathsf{vk}_C)$ and address A_C. Bob owns a chat key pair $(\mathsf{sk}_B^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}})$. The wallet key pairs are produced by calling KGen and the chat key pairs are the result of $\mathsf{KGenChat}$. The scenario is shown in Fig. 6.

A. A Naive Approach

The basic idea of our chat protocol follows [31] in embedding the hidden message amsg in the nonce of the ECDSA signatures. But while [31] only allows unidirectional messaging, our protocol allows for bidirectional communication. To create the bidirectional subliminal channel, we propose a new way

Security game $\mathsf{G}_{\ell,\mathsf{SIG},\mathsf{Chat}}$

Challenger C | **Attacker** $\mathcal{A}$

$(\mathsf{pk}_A^{\mathsf{Chat}}, \mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}, \mathsf{sk}_B^{\mathsf{Chat}}) \leftarrow\$ \mathsf{KGenChat}()$

for $i = 1, \ldots, \ell$:

$(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow\$ \mathsf{KGen}()$

$\mathsf{pk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}, (\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell) \rightarrow$

$b \leftarrow\$ \{0, 1\}$

for $i = 1, \ldots, \ell$

messages $\mathsf{msg}_i^{(1)}$, $\mathsf{msg}_i^{(2)}$, $\mathsf{amsg}_i^{(1)}$, $\mathsf{amsg}_i^{(2)}$ $\leftarrow$

if $b = 0$: **for** $j \in \{1, 2\}$:

$\mathsf{data}_{\mathsf{Chat}} := (\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}, \mathsf{amsg}_i^{(j)}, \mathsf{vk}_i)$

$\sigma_i^{(j)} \leftarrow\$ \mathsf{SignChat}_{E,G,n}(\mathsf{sk}_i, \mathsf{msg}_i^{(j)}, \mathsf{data}_{\mathsf{Chat}})$

if $b = 1$: **for** $j \in \{1, 2\}$:

$\sigma_i^{(j)} \leftarrow\$ \mathsf{Sign}_{E,G,n}(\mathsf{sk}_i, \mathsf{msg}_i^{(j)})$

$\sigma_i^{(1)}, \sigma_i^{(2)} \rightarrow$

end loop

bit b' $\leftarrow$

if $\mathsf{msg}_i^{(1)} \neq \mathsf{msg}_i^{(2)}$ for all $i = 1, \ldots, \ell$:

return $[b = b']$

else : **return** 0

Figure 5. Security game to model undetectability. The attacker chooses messages to sign and a message to embed. Their goal is to detect whether the provided signatures contain an embedded message or not.

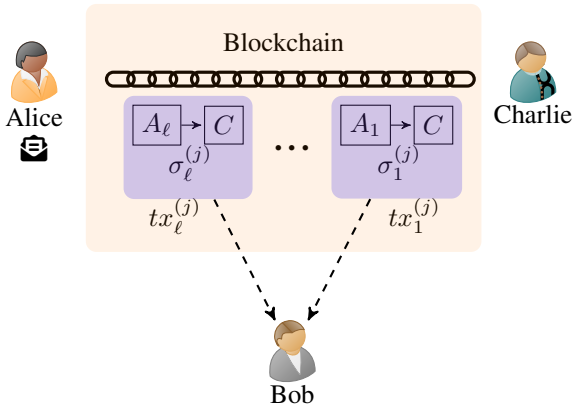

Figure 6. Subliminal chat from Alice to Bob: Alice signs and publishes transactions $tx_i^{(j)}$ for $i = 1, \ldots, \ell$ and $j = 1, 2$ on the blockchain. Bob analyzes the signatures and detects messages from Alice. Note that Charlie's identity is irrelevant for the chat and could be anyone, including Alice.

of leaking the secret signing key: Perform a non-interactive key exchange using the asymmetric key pairs generated by $\mathsf{KGenChat}$, which results in a shared secret that is used to derive a nonce during signing an ECDSA signature to leak the secret signing key. We call this new approach $\mathsf{NonceGenBasic}$. Its pseudocode is given in Fig. 7. The approach has several benefits compared to a pre-shared nonce:

- The nonce cannot be detected by binary analysis as it can be computed on-the-fly.
- If a nonce still becomes public knowledge, only communication between the two parties using it is endangered.
- The subliminal channel is independent of Alice's and Bob's Bitcoin addresses and keys. Therefore, Alice and Bob do not need to know each others Bitcoin addresses in advance in order to communicate.

$\mathsf{NonceGenBasic}(\mathsf{amsg}, \mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}})$

1 : $\mathsf{k}^{\mathsf{Chat}} := \mathsf{ECDH}(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}})$

2 : **if** $\mathsf{amsg} = \varnothing$

3 : **return** $\mathsf{k}^{\mathsf{Chat}}$

4 : **return** $\mathsf{Enc}(\mathsf{amsg}, \mathsf{k}^{\mathsf{Chat}})$

Figure 7. Basic idea for altering the nonce generation algorithm in order to embed chat messages in ECDSA signatures.

Obviously, using this naive approach is highly susceptible to resulting in a nonce reuse during signature generation which would undermine the security of ECDSA and is also easily detectable for our attacker $\mathcal{A}$. We will overcome these issues in the following section which concludes with an undetectability and security proof for the final nonce generation function.

B. Theory

Alice and Bob need to exchange their public chat keys in a way that ensures authenticity and integrity. How to realize this in practice will be discussed later and is out of scope of our theoretic model. To send a message amsg of b bytes to Bob, Alice splits amsg into $\ell = \lceil \frac{b}{32} \rceil$ parts $\mathsf{amsg}_1, \ldots, \mathsf{amsg}_\ell$. Also from now on we assume the length of amsg to be a multiple of 32 bytes. If this should not be the case, we simply append null bytes to amsg. To embed amsg_i, Alice then creates transactions $tx^{(1)}_{A_i,C}$ and $tx^{(2)}_{A_i,C}$. Whether the receiving addresses in the transactions are equal or not does not impact the sending of the message. But let Charlie's address A_C be the receiver in all transactions for simplicity.

Alice signs all transactions from wallet i with her signing key sk_i using the SignChat routine which makes use of NonceGenChat (cf. Fig. 8). She embeds her secret message part amsg_i into the transaction $tx^{(1)}_{A_i,C}$ by passing it to the signing routine via $\mathsf{data}_{\mathsf{Chat}} = (\mathsf{sk}^{\mathsf{Chat}}_A, \mathsf{pk}^{\mathsf{Chat}}_B, \mathsf{amsg}_i, \mathsf{vk}_i)$. Per call, NonceGenChat computes a shared secret $\mathsf{k}^{\mathsf{Chat}}_i = \mathsf{H}(\mathsf{ECDH}(\mathsf{sk}^{\mathsf{Chat}}_A, \mathsf{pk}^{\mathsf{Chat}}_B) \| \mathsf{vk}_i)$ and encrypts the current message block amsg_i with it. The resulting ciphertext is returned to be used as nonce by SignChat. We require the cipher to produce pseudo-random ciphertexts[5]. This is important to ensure that the ciphertext is suitable to be used as a nonce for the signature scheme. Because AES is believed to be a (t, ϵ, q)-pseudorandom permutation [57], AES-CBC produces (t, ϵ, q)-pseudorandom ciphertexts and is therefore suitable for our implementation. More concretely, this means that the ciphertexts produced by AES-CBC are (t, ϵ, q)-indistinguishable from uniformly distributed strings (upon random choice of the symmetric AES-key and adversarially chosen messages).

We concatenate the hash of the message to be signed ($tx^{(1)}_{A_i,C}$ in this case) with $\mathsf{k}^{\mathsf{Chat}}_i$ and use the first 128 bits of its hash as initialization vector. This ensures fresh nonces for each signature even if amsg contains two equal 32 byte blocks.

NonceGenChat is a deterministic algorithm. However, the nonce returned by NonceGenChat may lead to either r or s being zero during the signing process. In this case, SignChat can request a different nonce from NonceGenChat by incrementing the cnt variable which leads to $\mathsf{k}^{\mathsf{Chat}}$ being hashed cnt times before being used or embedded. Please note that the hashing of $\mathsf{k}^{\mathsf{Chat}}$ is only implemented to ensure the theoretic security of the signature scheme. A nonce causing r or s being zero only occurs in two out of 2^{256} cases which means that the additional hashing does not occur in practice.

When signing the second transaction $tx^{(2)}_{A_i,C}$, Alice now chooses $\mathsf{data}_{\mathsf{Chat}} = (\mathsf{sk}^{\mathsf{Chat}}_A, \mathsf{pk}^{\mathsf{Chat}}_B, \varnothing, \mathsf{vk}_i)$ which tells the nonce generation function to embed $\mathsf{k}^{\mathsf{Chat}}_i$ instead of a message. This way, the shared secret $\mathsf{k}^{\mathsf{Chat}}_i$ itself is used as nonce which allows Bob to later recover sk_i. Alice has to ensure to leak sk_i only once per wallet as otherwise a nonce reuse occurs which would allow anyone monitoring the blockchain to take over their wallet.

After all transactions and signatures are generated, Alice publishes all transactions to the blockchain. In cases where Bob is assumed to be malicious, Alice has to take extra care when publishing $tx^{(2)}_{A_i,C}$ as this transaction allows Bob to recover the private signing key sk_i for address A_i. Therefore, all other transactions originating from A_i must be mined into a block in the longest chain and only a transaction fee should be left on A_i before publishing $tx^{(2)}_{A_i,C}$.

Bob on the other end follows the extraction algorithm depicted in Fig. 9 to receive amsg_i. He can compute $\mathsf{k}^{\mathsf{Chat}}_i$ from $\mathsf{sk}^{\mathsf{Chat}}_B$, $\mathsf{pk}^{\mathsf{Chat}}_A$ and vk_i. Bob searches the blockchain for a transaction $(tx^{(2)}_{A_i,C}, \sigma^{(2)})$, which can be identified by r as $r^{(2)}$ is the x-coordinate of the point $\mathsf{k}^{\mathsf{Chat}}_i \cdot G$. In theory, an additional check for r being the x-coordinate of G multiplied with a hashed $\mathsf{k}^{\mathsf{Chat}}_i$ would be necessary. But since $\mathsf{k}^{\mathsf{Chat}}_i$ does not get hashed in practice[6] we discarded this test for simplicity. Because Bob knows the nonce used for signing $tx^{(2)}_{A_i,C}$, he can compute sk_i from either $(r^{(2)}, s^{(2)})$ or $(r^{(2)}, -s^{(2)})$. Decision for the correct sk_i can be made by computing the corresponding verification key and comparing it with vk_i. This step is necessary because Bitcoin only accepts normalized signatures. Next, Bob searches the blockchain for the remaining transaction $(tx^{(1)}_{A_i,C}, \sigma^{(1)})$. Since Bob learned

$\mathsf{NonceGenChat}(\mathsf{H}(\mathsf{msg}), \mathsf{data}_{\mathsf{Chat}}, cnt)$

1 : $(\mathsf{sk}^{\mathsf{Chat}}_A, \mathsf{pk}^{\mathsf{Chat}}_B, \mathsf{amsg}, \mathsf{vk}_i) := \mathsf{data}_{\mathsf{Chat}}$
2 : $\mathsf{k}^{\mathsf{Chat}}_i \leftarrow \mathsf{H}(\mathsf{ECDH}(\mathsf{sk}^{\mathsf{Chat}}_A, \mathsf{pk}^{\mathsf{Chat}}_B) \| \mathsf{vk}_i)$
3 : **for** $j = 1 \ldots cnt$:
4 : $\quad \mathsf{k}^{\mathsf{Chat}}_i \leftarrow \mathsf{H}(\mathsf{k}^{\mathsf{Chat}}_i)$
5 : **if** $\mathsf{amsg} = \varnothing$:
6 : $\quad$ **return** $\mathsf{k}^{\mathsf{Chat}}_i$
7 : $iv := \mathsf{H}(\mathsf{H}(\mathsf{msg}) \| \mathsf{k}^{\mathsf{Chat}})[0:128]$
8 : $\mathsf{ctx}_{\mathsf{Chat}} := \mathsf{AES}^{\mathsf{Enc}}_{\mathsf{CBC}}(\mathsf{amsg}, \mathsf{k}^{\mathsf{Chat}}_i, iv)$
9 : **return** $\mathsf{ctx}_{\mathsf{Chat}}$

$\mathsf{SignChat}_{E,G,n}(\mathsf{sk}_i, \mathsf{msg}, \mathsf{data}_{\mathsf{Chat}})$

1 : $c \leftarrow 0$
2 : $h := \mathsf{H}(\mathsf{msg})$
3 : $k \leftarrow \mathsf{NonceGenChat}(h, \mathsf{data}_{\mathsf{Chat}}, c)$
4 : $(x, y) := k \cdot G$
5 : $r := x \bmod n$
6 : **if** $r = 0$:
7 : $\quad c \leftarrow c + 1$; goto line 3
8 : $s := [k^{-1}(h + r \cdot \mathsf{sk}_i)] \bmod n$
9 : **if** $s = 0$:
10 : $\quad c \leftarrow c + 1$; goto line 3
11 : **return** $(r, \min\{s, -s\})$

Figure 8. Nonce generation algorithm for ECDSA signatures that enables the user to efficiently embed secret messages into the signatures in a provably undetectable way. As in SignECDSA, the message msg to be signed is the transaction itself. The variable $\mathsf{data}_{\mathsf{Chat}}$ contains all additionally needed information for the embedding.

[5]Note: This is a stronger assumption than requiring indistinguishability of ciphertexts.

[6]Recall, NonceGenChat hashes $\mathsf{k}^{\mathsf{Chat}}_i$ with a probability of 2^{-255}.

sk_i from $(tx^{(2)}_{A_i,C}, \sigma^{(2)})$, he can compute all the nonces used for signing $tx^{(1)}_{A_i,C}$ and hence extract the encrypted message which is either $\mathsf{ctx}_{\mathsf{Chat}}$ from $(r^{(1)}, s^{(1)})$ or $\mathsf{ctx}'_{\mathsf{Chat}}$ from $(r^{(1)}, -s^{(1)})$. Only one of the two ciphertexts results in a meaningful plaintext when decrypted with $\mathsf{k}_i^{\mathsf{Chat}}$, so Bob is able to recover amsg_i successfully. In our PoC implementation we define "meaningful" as "ASCII-printable". This is no hard constraint as binary data can be Base64 encoded before sending. Note that each transaction in Bitcoin has a unique timestamp that is determined at the time the transaction is broadcasted to the blockchain. If Alice sends the message part amsg_i before amsg_{i+1}, Bob can determine the correct ordering of the message parts by looking at these timestamps.

Lemma 1: If the $(t, \epsilon, 2)$-DDH assumption is true for secp256k1 and AES is a $(t, \epsilon, 1)$-pseudorandom permutation, then the output of the algorithm $\mathsf{NonceGenChat}(\mathsf{H}(tx), \mathsf{data}_{\mathsf{Chat}}, cnt)$ is $(t, 2\epsilon + 2^{-127}, 1)$-indistinguishable from the uniform distribution on $\{1, \ldots, n-1\}$ for a distinguisher that does not know $\mathsf{data}_{\mathsf{Chat}}$. Here, n is the order of secp256k1.

Proof: Consider the different games presented in Fig. 10 in the appendix. Here, G_1 equals NonceGenChat, while G_7 corresponds to the choice of a random nonce.

$\mathsf{G}_1 \approx \mathsf{G}_2$: As we assume the $(t, \epsilon, 2)$-decisional Diffie-Hellman assumption for secp256k1, no attacker can distinguish the output of $\mathsf{ECDH}(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}})$ from a randomly chosen group element of secp256k1.

$\mathsf{G}_2 = \mathsf{G}_3$: As we work in the random-oracle model and the random element g is not known by the adversary, the output $\mathsf{k}_i^{\mathsf{Chat}}$ of $\mathsf{H}(g \| \mathsf{vk}_i)$ is a uniformly distributed string of length 256. Similarly, applying H on $\mathsf{H}(tx) \| \mathsf{k}_i^{\mathsf{Chat}}$ and shortening to the first 128 bits gives a uniformly distributed string of length 128.

$\mathsf{G}_3 = \mathsf{G}_4$: Applying a random oracle H to a uniformly random value results again in a uniformly random value. We can thus ignore the for-loop.

$\mathsf{G}_4 \approx \mathsf{G}_5$: As we assume that AES is a $(t, \epsilon, 1)$-pseudorandom permutation and $\mathsf{AES}^{\mathsf{Enc}}_{\mathsf{CBC}}$ is called with a secret key $\mathsf{k}_i^{\mathsf{Chat}}$ and a randomly chosen initialization vector iv, the generated ciphertext $\mathsf{ctx}_{\mathsf{Chat}}$ are $(t, \epsilon, 1)$-pseudorandom and thus indistinguishable from a uniformly sampled string of length 256.

$\mathsf{G}_5 = \mathsf{G}_6$: The value of iv is not used in the algorithm anymore. Furthermore, the only part of $\mathsf{data}_{\mathsf{Chat}}$ still used is amsg_i. But amsg_i is used only in the if-statement and in both cases ($\mathsf{amsg}_i = \varnothing$ or $\mathsf{amsg}_i \neq \varnothing$), the output is a uniformly random string of length 256.

$\mathsf{G}_6 \approx \mathsf{G}_7$: The probability that $\mathsf{ctx}_{\mathsf{Chat}}$ is not in the interval $\{1, \ldots, n-1\}$ is at most $1 - \frac{n-1}{2^{256}}$. As $2^{256} - n \leq 2^{129}$, this probability is bounded by 2^{-127}, which is negligible[7].

Combining the security losses in $\mathsf{G}_1 \approx \mathsf{G}_2$ (by the DDH), in $\mathsf{G}_4 \approx \mathsf{G}_5$ (by the PRP), and in $\mathsf{G}_6 \approx \mathsf{G}_7$ (by the order of secp256k1), we can conclude that $\mathsf{NonceGenChat}(\mathsf{H}(tx), \mathsf{data}_{\mathsf{Chat}}, cnt)$ is $(t, 2\epsilon + 2^{-127}, 1)$-indistinguishable from the uniform distribution on $\{0,1\}^{256}$. ■

[7]Note that $n = 2^{256} - 432420386565659656852420866394968145599$ for secp256k1.

$\mathsf{ExtChat}_{E,G,n}(\mathsf{sk}_B^{\mathsf{Chat}}, \mathsf{pk}_A^{\mathsf{Chat}}, A_C)$

$Tx_C \leftarrow$ all (tx, σ) sending BTC to A_C; $i \leftarrow 1$

foreach (tx, σ) **in** Tx_C :

$\mathsf{vk}_i, A_i, \mathsf{k}_i^{\mathsf{Chat}} :=$ $\mathsf{hasAmsg}(E, G, n, \mathsf{sk}_B^{\mathsf{Chat}}, \mathsf{pk}_A^{\mathsf{Chat}}, tx, \sigma)$

1 : get vk_i from tx
2 : $(r, s) := \sigma$
3 : $\mathsf{k}_i^{\mathsf{Chat}} := \mathsf{H}(\mathsf{ECDH}(\mathsf{sk}_B^{\mathsf{Chat}}, \mathsf{pk}_A^{\mathsf{Chat}}) \| \mathsf{vk}_i)$
4 : $(x, y) := \mathsf{k}_i^{\mathsf{Chat}} \cdot G$
5 : **if** $r = x \bmod n$:
6 : **return** $\mathsf{vk}_i, \mathsf{Addr}(\mathsf{vk}_i), \mathsf{k}_i^{\mathsf{Chat}}$
7 : **return** $0, 0, 0$

if $\mathsf{vk}_i = 0$ or $\mathsf{Vf}(\mathsf{vk}_i, tx, \sigma) = 0$: continue

$\mathsf{sk}_i \leftarrow$ $\mathsf{recoverSk}(E, G, n, \mathsf{k}_i^{\mathsf{Chat}}, \mathsf{vk}_i, tx, \sigma)$

1 : $h := \mathsf{H}(tx)$; $(r, s) := \sigma$
2 : $\mathsf{sk}_i \leftarrow [(s \cdot \mathsf{k}_i^{\mathsf{Chat}} - h) \cdot r^{-1}] \bmod n$
3 : **if** $\mathsf{sk}_i \cdot G = \mathsf{vk}_i$:
4 : **return** sk_i
5 : $\mathsf{sk}_i \leftarrow [((-s) \cdot \mathsf{k}_i^{\mathsf{Chat}} - h) \cdot r^{-1}] \bmod n$
6 : **if** $\mathsf{sk}_i \cdot G = \mathsf{vk}_i$:
7 : **return** sk_i
8 : **return** 0

if $\mathsf{sk}_i = 0$: continue

$Tx_{A,C} \leftarrow$ all $(tx_{A,C}, \sigma)$ where $tx_{A,C} \neq tx$

$\mathsf{amsg}_i \leftarrow$ $\mathsf{recoverMsg}(n, \mathsf{k}_i^{\mathsf{Chat}}, \mathsf{sk}_i, Tx_{A,C})$

1 : **foreach** $(tx, (r, s))$ **in** $Tx_{A,C}$:
2 : $h := \mathsf{H}(tx)$; $iv := \mathsf{H}(h \| \mathsf{k}_i^{\mathsf{Chat}})[0:128]$
3 : $\mathsf{ctx}_{\mathsf{Chat}} \leftarrow [(\mathsf{sk}_i \cdot r + h) \cdot s^{-1}] \bmod n$
4 : $tmp \leftarrow \mathsf{AES}^{\mathsf{Dec}}_{\mathsf{CBC}}(\mathsf{ctx}_{\mathsf{Chat}}, \mathsf{k}_i^{\mathsf{Chat}}, iv)$
5 : **if** tmp is meaningful :
6 : **return** tmp
7 : $\mathsf{ctx}_{\mathsf{Chat}} \leftarrow [(\mathsf{sk}_i \cdot r + h) \cdot (-s)^{-1}] \bmod n$
8 : $tmp \leftarrow \mathsf{AES}^{\mathsf{Dec}}_{\mathsf{CBC}}(\mathsf{ctx}_{\mathsf{Chat}}, \mathsf{k}_i^{\mathsf{Chat}}, iv)$
9 : **if** tmp is meaningful :
10 : **return** tmp

$i \leftarrow i + 1$

reconstruct amsg from $(\mathsf{amsg}_i)_{i=1,\ldots,\ell}$

Figure 9. Extraction algorithm used to recover messages that were embedded into ECDSA signatures with the SignChat algorithm. First, the extractor iterates through all transactions and determines whether they contain embedded messages via hasAmsg. Then, the corresponding signing keys are recovered via recoverSk and finally, recoverMsg recovers the messages.

The above lemma implies that the nonce generated by algorithm NonceGenChat looks like a random nonce. But, in order to construct an undetectable chat system, we must guarantee that the nonces are indistinguishable from the deterministic

nonces generated by NonceGenRFC6979. Note that if every message is signed at most once (which is true for Bitcoin), this holds, as the distinguisher does not know the signing key sk_i.

Lemma 2: Algorithm $\mathsf{NonceGenRFC6979}(\mathsf{H}(\mathsf{msg}), d, cnt)$ outputs are $(t, 0, 1)$-indistinguishable from the uniform distribution on the set $\{1, \ldots, n-1\}$ in the random oracle model for every t. Here n is the order of secp256k1.

Combining these two lemmata allows us to conclude the undetectability of our chat client.

Theorem 1: If the $(t, \epsilon, 2)$-DDH assumption is true for secp256k1, AES is a $(t, \epsilon, 1)$-pseudorandom permutation, and ECDSA is $(t, \epsilon, 2)$-secure, then our client Chat is $(t, 3\epsilon + 2^{-127})$-undetectable for $\ell \leq t$ messages (with respect to ECDSA) in the random oracle model.

Proof: Lemma 1 shows that under the given assumptions, the generated nonces of $\mathsf{NonceGenChat}(\mathsf{H}(tx), \mathsf{data}_{\mathsf{Chat}}, cnt)$ are $(t, 2\epsilon + 2^{-127}, 1)$-indistinguishable from random nonces and Lemma 2 shows that this is in turn indistinguishable from the output of $\mathsf{NonceGenRFC6979}(\mathsf{H}(\mathsf{msg}), d, cnt)$. Hence, the generated nonces are $(t, 2\epsilon + 2^{-127}, 1)$-indistinguishable from real nonces. The $(t, \epsilon, 2)$-security of ECDSA now implies that an attacker cannot extract the nonces from the signatures. These nonces are thus not known by an attacker and the remaining part of SignChat are identical to ECDSA. Hence, Chat is $(t, 3\epsilon + 2^{-127})$-undetectable for $\ell \leq t$ messages (with respect to ECDSA) in the random oracle model. ∎

With the chat being provably undetectable, we do not need to care for confidentiality of the chat messages being sent. But an attacker may still try to attack the integrity of embedded messages by altering transactions that are broadcast to the network in the hope of hitting a transaction with an embedded message. This attack, however, has to fail as altering a transaction renders the transaction signature invalid. As a result, all blockchain nodes will reject and not further broadcast the transaction. Therefore, embedded messages that are tampered with will automatically be filtered out by the network. Bob still verifies the transaction signatures in cases where the Bitcoin node forwarding the transaction to him cannot be trusted. Authenticity of the messages is guaranteed as only Alice owns $\mathsf{sk}_A^{\mathsf{Chat}}$ which means that only she can encrypt a message with $\mathsf{k}_i^{\mathsf{Chat}}$.

Last but not least we want to stress that the communication flow between Alice and Bob is independent of the flow of the bitcoins transferred during communication. Alice can hide messages meant for Bob in any of her transactions, no matter the recipient. As a result, no *metadata* of the communication between Alice and Bob is generated or observable by third parties. This is possible because the peer-to-peer network structure is used for transaction and therefore message distribution. Also, Bob could receive messages without knowing the mailbox address (A_C in this case) by simply scanning all transactions in new blocks for messages from Alice. So changing the mailbox address while communicating is possible.

C. Considering multi-input transactions

So far we proposed and demonstrated a technique to securely send subliminal messages via Bitcoin transaction. Our design, as well as other work like [31] require an overhead of one transaction to leak the signing key of the sender. Also, our technique requires the sender to issue multiple requests from the same Bitcoin address. Since it is recommended to use each address only once to ensure the user's privacy, our "chat behavior" might attract some attention.

We can overcome both issues with a feature of cryptocurrencies based on the UTXO model[8] like Bitcoin, Litecoin, or Dash: multi-input transactions. UTXO model based cryptocurrencies allow[9] transactions to have multiple inputs. Since each input contains a signature, we can use this to send arbitrarily long messages with just a single transaction. We only require that one of the inputs is signed using $\mathsf{k}^{\mathsf{Chat}}$ as nonce. All other inputs are then signed using ciphertexts as nonces.

This improvement cannot be applied to account based cryptocurrencies like Ethereum or XRP. Here, multiple transactions have to be used in order to transmit a hidden message. But since sending multiple transactions from the same account is common for account-based cryptocurrencies, sending a message via multiple transactions will not raise any suspicion.

D. PoC implementation

We implemented a proof of concept (PoC)[10] of the chat client in Python. The program relies on the Python framework *bit*[11] to interact with the blockchain and manage keys. For key generation, signing, and verification, bit relies on *coincurve*[12] which is a Python wrapper for the highly optimized *libsecp256k1*. libsecp256k1 is a good choice for our experiments because of three reasons: First, it is part of the Bitcoin reference implementation *bitcoin-core*. This implies that all changes we do to the library are applicable to other compliant libraries as well. Second, libsecp256k1 provides a very handy API, allowing us to provide the signing routine with a custom nonce generation function that can be passed, besides the transaction hash and the private key, arbitrary additional data. Instead of passing a custom nonce generation function to the signing routine, one can also easily exchange the default nonce generation function with our handcrafted routine. Third, libsecp256k1 implements ECDH support. This means that the number of code changes for ECDH support is close to zero. We only have to implement AES and define the $\mathsf{data}_{\mathsf{Chat}}$ struct. Note that curve *secp256k1* is also used by Ethereum, XRP, Litecoin, EOS, and several other top 100 cryptocurrencies [52].

To implement AES, we rely on Intel's reference implementation for AES-NI as given in [58]. The code provides an efficient AES implementation while keeping the code base small and avoiding S-boxes or T-tables which would blow up the code base. We use the ECDH and SHA256 implementation provided by libsecp256k1 to compute $\mathsf{k}^{\mathsf{Chat}}$. We define $\mathsf{data}_{\mathsf{Chat}}$ in the main header file of libsecp256k1 and modify coincurve and bit to support it.

[8] Cryptocurrencies based on the UTXO model require the user to always spend inputs as a whole and optionally return some coins as change.

[9] Due to their design, they even *require* multi-input transactions.

[10] https://github.com/7K9cNgkh/btc-chat

[11] https://github.com/ofek/bit

[12] https://github.com/ofek/coincurve

Table II
CHAT POC BETWEEN ALICE AND BOB ON THE BITCOIN TESTNET3 USING MULTIPLE INPUTS TO HIDE THE MESSAGE.

Alice	BTC Priv. Key	sk_A	27ded4e06a96cc5613e0e2bfd48ab068cfa0ab687c5900cad4e4c2d79b37794d
	BTC Verif. Key	vk_A	0386aaca9192ae7434a677ffb4774dac8e8e9527757f3af6e812a290b8627a91ab
	BTC Addr.	A_A	mpiNLrkV6DpQKDW3oPQm1RXhJ6azf43o5m
	Chat Priv. Key	$\mathsf{sk}_A^{\mathsf{Chat}}$	89bdcbeb3e1dddc3eabf472b431f9e41b6e5ed327d1166f9aa523112237a0162
	Chat Pub. Key	$\mathsf{pk}_A^{\mathsf{Chat}}$	032f84f427dda0a6f5b8b78e569b2aa213f030ec36db295a1934eb4440587cfdce
Bob	BTC Priv. Key	sk_B	38147cca2c3a120c6525f45aaa51d6ad624ed06ea67a0b8f285b575ca0359792
	BTC Pub. Key	vk_B	03ee8f3721316cd7457b141ceeb42e26b0f72c24fcd14ae96ade6ebd2f6d180dd1
	BTC Addr.	A_B	mzvoy8wgtuDB3ng7Ga4758fNPrsFch8mLr
	Chat Priv. Key	$\mathsf{sk}_B^{\mathsf{Chat}}$	935cece8609c40ecb802e9c496ea99db9f8da468a264ab1ebb65eb13d5bcebad
	Chat Pub. Key	$\mathsf{pk}_B^{\mathsf{Chat}}$	02e8a27b4f01a53e69b80495a955ed2e974290ff3b862e542ac1d92aa0dc7a39fa
A → B	Message	amsg	"Hi Bob, how are you?" (20 Bytes)
	Transaction ID	$tx_{A\to B}$	b282f4f2298f197b2abd848c5da4dc83dd88851e714612785237457f18dab378
B → A	Message	amsg	"I'm great! This chat is nice. We're hiding in the open." (55 Bytes)
	Transaction ID	$tx_{B\to A}$	fc22fb96908c2c9730b796c0c346e717564e3bc6b63b33eeae8d05e7fd97093a

In order to demonstrate the PoC we placed two transactions $tx_{A\to B}$ and $tx_{B\to A}$ with embedded messages on the Bitcoin Testnet3 blockchain. Here, we denote by $tx_{X\to Y}$ the fact that a message is sent from X to Y in a transaction sent from X to some third party. All necessary information to receive the transferred messages yourself is given in Table II. Both transactions send bitcoins to an arbitrary letterbox address while exchanging messages between Alice and Bob. The transaction $tx_{A\to B}$ has two inputs embedding a 20 byte message from Alice to Bob. The signature of the first input carries the encrypted message while the signature for the second input is used to leak sk_A to Bob. The transaction $tx_{B\to A}$ contains a 55 byte message from Bob to Alice. Because the message exceeds 32 bytes, a third input is needed. The first two input signatures contain the message and the third signature leaks sk_B to Alice. Both transactions transfer all remaining funds to a new wallet.

Performance Benchmarking the PoC shows that our approach is practical. The benchmarks we collected are depicted in Table III. We measured the performance for the unmodified and the chat version of libsecp256k1. The usage of AES or additional hash operations does not add significant time penalties. The main timing difference of about $50\mu s$ between the unmodified library and the chat libsecp256k1 is caused by the additional group operation during the offline key exchange.

The data throughput is upper-bounded by the capacity of the blockchain. Currently Bitcoin issues one block with approx. 2.2k transactions every 10 minutes. Assuming each transaction has 1.6 inputs (on average, cf. Table IV) and each input carries 32 B of information, the upper bound for data transfer of our approach in Bitcoin is about 112.6 kB every ten minutes or 187.7 B per second. Later blockchains like EOS, or XRP already offer a much higher capacity as they are capable of mining thousands of transactions per second [59]. Future blockchain improvements may even allow for 100.000 transactions per second [60].

Table III
BENCHMARKS. ALL VALUES ARE AVERAGED OVER 1000000 RUNS ON AN INTEL CORE I5-7600 CPU.

Algorithm	min (μs)	avg (μs)	max (μs)
AES-NI-CBC ENC (w/ KE)	0.0612	0.0631	0.0691
AES-NI-CBC DEC (w/ KE)	0.0611	0.0622	0.0639
SHA256	0.525	0.529	0.546
ECDH	49.2	50.2	51.9
NonceGenRFC6979	6.22	6.37	6.56
NonceGenChat	55.2	56.7	58.1
SignBTC	41.5	42.4	43.5
SignChat	90.9	92.9	93.9

E. Practical considerations for a real implementation

Before using our PoC in a practical setting, some additional considerations should be taken into account.

Exchanging public chat keys When Alice and Bob want to communicate, they initially have to exchange each others public chat keys $\mathsf{pk}_A^{\mathsf{Chat}}$ and $\mathsf{pk}_B^{\mathsf{Chat}}$. To do so, they can rely on a public key infrastructure (PKI) [61] as we do not require the keys to be kept secret[13]. Any form of communication that ensures authenticity and integrity is suitable for this initial step as well. In cases where the pure fact that Alice and Bob exchange some keys is not to be known by an attacker, however, more sophisticated methods or a personal meeting may be required.

Choice of the mailbox address The choice of the mailbox address is worth a thought. For sure, choosing a mailbox address that is owned by one of the communication partners will allow for efficiently receiving messages as the number of transactions received by the mailbox address and therefore the number of transactions that have to be searched for hidden messages will be at a minimum. On the other hand, a random address receiving many transactions (due to chatting) may raise suspicion in the context of transaction pattern analysis. Therefore, it might be wise to agree on multiple mailbox

[13] In fact, we even assume that $\mathsf{pk}_A^{\mathsf{Chat}}$ and $\mathsf{pk}_B^{\mathsf{Chat}}$ are known to the attacker.

Table IV
DISTRIBUTION OF TRANSACTION INPUTS PER BITCOIN TRANSACTION MINED BETWEEN FEB. 22. AND FEB. 28., 2021.

#inputs	1	2	3	4	5	6+
percentage	74.29%	13.62%	4.28%	2.02%	1.25%	4.54%

addresses and switch between them or choosing a well known mailbox address assumed to receive many transactions like addresses belonging to cryptocurrency exchanges or charity.

Also, Alice may embed messages into regular transactions that are about to occur anyways. In this case Bob does not know the mailbox address Alice sent the transaction to, so he has to scan all transactions on the blockchain for new messages. During initial setup, all transactions mined into blocks after Alice and Bob exchanged their public chat keys have to be searched. Afterward, only transactions in new blocks have to be searched. For slower blockchains this is feasible as e. g. new Bitcoin blocks get mined every 8-15 minutes with 1500-2500 transactions each [62], [63]. The faster the blockchain is, the less efficient becomes the message receiving algorithm for unknown mailbox addresses while allowing for a maximum protection against transaction pattern analysis.

Choice of the blockchain When implementing our approach in a real chat application, the choice of the underlying blockchain is important. If the application is only meant to be used by a small amount of users and only for critical but short messages, the Bitcoin blockchain might be suitable although it only allows for a throughput of 187 bytes per second and requires varying transaction fees of 2 - 60 US dollar per transaction [64]. For applications with many users, the chosen blockchain should be capable of mining many transactions per second. For an example, EOS and XRP are capable of handling thousands [59] and Ethereum 2.0 even promises 100.000 transactions per second [60]. While, this will still not be enough to handle services equivalent to WhatsApp which deals with about 1.2m messages per second [65], a reasonable chat application for those who really need it can be realized. Besides the throughput, the transaction fee is important as well. As messages are embedded into transactions, a fee charged per transaction is also charged per message. Therefore, blockchains with little or no transaction fees allow cheaper messaging. XRP, as an example, charges a fraction of a US cent per transaction [64].

Multiple transactions vs. multi-input transactions We analyzed all 2,131,422 Bitcoin transactions within a week[14] (blocks 671623-672620) for their number of inputs and found that 25.71% of the transactions had more than one input. In general, it became obvious that transactions with more inputs are less likely to occur. A true chat client implementation should take this distribution into account. This means that in practice, a balance between sending messages in multi-input transactions and multiple transactions should be chosen to keep the communication hidden.

[14]February 22. - 28., 2021

Dealing with a malicious receiver If Bob is assumed to be malicious, Alice has to take extra care of her coins. As soon as Alice publishes a transaction with a signature that uses $\mathsf{k}^{\mathsf{Chat}}$ as nonce Bob can compute Alice's private signature key sk_A and therefore owns the wallet. Hence Bob is able to generate valid transactions for that wallet and can thus spend any remaining funds. This may even be true for the funds being transferred in the very transaction that leaks sk_A to Bob if the blockchain uses the UTXO model. This is because Bob may learn the transaction before it is mined into a block. If Bob computes sk_A and issues a new transaction spending the same UTXOs again but raises the transaction fee, then it is more likely that Bob's new transaction gets mined and Alice's transaction is discarded. To overcome this problem, only small funds should be transferred in the transaction leaking sk_A and no funds should remain in the wallet afterwards. This may be achieved by sending a message using a multi-input transaction where no signature leaks sk_A and all but little coins are transferred to a save wallet. After the transaction is mined into a block, Alice may publish the transaction spending the remaining funds and leaking sk_A. Now Bob can read the message contained in the previous transaction but has no motivation to betray Alice.

Forward secrecy and disappearing messages In cases where a UTXO-based blockchain is used and Bob is trustworthy, Alice and Bob can cooperate to achieve forward secrecy and disappearing messages. To do so, they can apply the cleaning scheme proposed in [66]. Here, Bob basically behaves just like in the malicious case described above: After computing sk_A, Bob issues a new transaction with a higher transaction fee that double-spends the UTXOs Alice used for leaking sk_A. The higher fee is important to increase the chance of Bob's transaction being mined into a block. This leads to the rejection of Alice's transaction because of double-spending. Therefore, an attacker cannot read the messages on the blockchain even though they might get hold of Alice's or Bob's chat key pair in the future because they cannot get a hold of sk_A anymore. Also, disappearing messages can be realized by having Bob double-spend all UTXOs used in transactions that Alice embeds messages in. In this case, Alice and Bob have to be online at the same time as the blockchain cannot be searched for new messages later on. However, we have to note that this cleaning scheme is not guaranteed to work as higher fees do not guarantee but only increase the probability that the double-spending transactions will be mined first.

V. RELATED WORK

Besides the works already mentioned in Section I, there have been several other studies with the goal on building covert channels in blockchain systems. For example, Basuki et al. [67] give a covert channel encoding scheme base on smart contracts with a joint use of image steganography. Abdulaziz et al. [68] create a decentralized messaging application utilizing Whisper – the communication protocol of Ethereum – to send encrypted messages both securely and anonymously. In [69], similarly as in [68] and [70], the authors use Whisper which relies on payload to store information useful for the realization of covert communication. All of these papers rely on specific

properties of the underlying blockchain and are therefore not easily adoptable to other blockchains. We circumvent this issue by embedding the messages in signatures generated by schemes that are widely used in blockchain systems.

Ali et al. [30] propose to use rejection sampling to embed messages in ECDSA signatures. Rejection sampling samples random nonces k until a pair (r, s) is found such that $\mathsf{PRF}_{\mathsf{k}^{\mathsf{Chat}}}((r, s)) = \mathsf{amsg}$ for some pseudorandom function PRF. It is known that this approach is undetectable (see e. g. [14], [3]), but has a very limited rate logarithmic in the length of the nonce. Partala [71] proposes a blockchain-based covert channel in which a transaction can carry one bit in the field of payments. While this scheme again is provably undetectable, its embedding rate is too low to be used in real applications. While Zhang et al. [72] improve the method of [71] using the special addresses generated by a Bitcoin address generator, the embedding rate is still too low to realize a chat. In contrast, our embedding scheme is provable secure while allowing for a high embedding rate with a constant overhead.

Some works also use Bitcoin's output script function `OP_RETURN` to directly embed messages in transactions [30], [66] or to identify transactions that embed messages [73]. While Ali et al. [30] do not describe how to encode messages for the script, Yin et al. [66] embed Base64-encoded ciphertexts using a symmetric encryption scheme. With high probability, both methods are detectable by statistical tests as they do not take the distribution of real `OP_RETURN` payloads into account. Therefore, these methods cannot be used in practice. Tian et al. [73] do take the distribution into account when generating identifiers for their transactions. As embedding technique, they replace the private signing key of the Bitcoin address with a ciphertext resulting from symmetrically encrypting amsg. To leak the private key d, they reuse the nonce k that is involved in the signing process. Leaking d through nonce reuse is also discussed by Ali et al. [30] and Frkat et al. [31] who both embed messages by replacing the nonce with a symmetrically computed ciphertext of amsg. Reusing the nonce results in the first part r of the signatures being equal which makes this technique easily detectable by an adversary. Note that it is in fact known that there are several parties scanning the blockchain for such weak signatures (see e. g. [50]). In addition to being detectable, an adversary can compute d as well and therefore impersonate the owner of the corresponding Bitcoin address. This is a major problem which renders this technique impractical.

To avoid the reuse of the nonce k, Frkat et al. [31] also propose a more involved method. The sender first symmetrically encrypts the messages $\mathsf{amsg} = \mathsf{amsg}_1, \ldots, \mathsf{amsg}_\ell$ using $\mathsf{k}^{\mathsf{Chat}}$ into ciphertexts $c_1, \ldots, c_\ell$ and uses these ciphertexts $c_1, \ldots, c_\ell$ as nonces in the production of the signature. Finally, to construct the signature $\ell + 1$, it uses $\mathsf{H}(\mathsf{k}^{\mathsf{Chat}})$ as nonce. The extractor stores all of the observed signatures and tries to reconstruct the private signing key d by using $\mathsf{H}(\mathsf{k}^{\mathsf{Chat}})$ as nonce applying the attacks described in Section II-D1. For the signature $\ell+1$, the extractor succeeds and thus reveals d. Using d, the extractor is able to reconstruct $c_1, \ldots, c_\ell$ and to decrypt them to the message amsg. As the attacker $\mathcal{A}$ can not observe the internal randomness used by the signing algorithm, all of the values $c_1, \ldots, c_\ell$, as well as $\mathsf{H}(\mathsf{k}^{\mathsf{Chat}})$ are indistinguishable from random nonces. This approach is the only one in the current literature that offers a reasonable bandwidth and might be secure. However, there are some drawbacks when using it for implementing a chat. First of all, the security of the embedding scheme remains unproven. Second, as Frkat et al. design their stegosystem to be used by botnets, communication is only possible in one direction while the reverse direction is declared out of scope. Also, sender and receiver have to agree on an initial symmetric key/nonce. In the context of botnets, this value can be hard-coded into the bot software, but for a chat application, this is rather unhandy. Each embedded message then has to contain the symmetric key that will be used as nonce k and for encrypting the next message. This effectively reduces the bandwidth of the channel. In contrast, our approach addresses all of these issues. We are provably undetectable while allowing for bidirectional communication and efficient key distribution without having to beforehand.

Another recent line of research, which is closely related to our work, deals with algorithm substitution attacks (ASA), and particularly with ASAs against digital signature schemes [14], [74], [75], [13]. Although the main focus of this paper are subliminal channels in digital signatures, we note that our approach can be used to realize an asymmetric ASA against ECDSA as well as against any splittable signature scheme (for a definition, see [75]). The resulting ASA would look similar to the attack given recently by Wang et al. [75], but would have the advantage to embed arbitrary messages, like e. g., sensitive data of users, and not just the secret signing key.

VI. Conclusion

In this work, we presented a new method to hide data in digital signatures and applied it to enable subliminal bidirectional communication in blockchain transactions in an asymmetric key scenario. We can hide messages in arbitrary transactions, and can thus apply our technique to arbitrary blockchains. The only requirement is the use of a splittable signature scheme for the transactions, such as the widely used ECDSA or EdDSA. Our subliminal communication channel follows a public-key approach and thus does not rely on hard-coded secrets as used for prior unidirectional proposals. Unlike classic secure messaging, where the *content* of messages is protected and private, but communication patterns or connection graphs are accessible to the operator of the messaging service, our scheme hides both the content and the mere existence of messages, thus leaving no metadata to be analyzed by other parties. We have shown that the channel is undetectable, meaning that both the content is secure and there is no externally observable metadata. Furthermore, the scheme features a low overhead of just one signature. To show that the protocol is practical, we implemented a proof-of-concept chat client for the Bitcoin blockchain and embedded chat messages in the Bitcoin Testnet3.

References

[1] G. J. Simmons, "The prisoners' problem and the subliminal channel," in *Advances in Cryptology – CRYPTO '83*, D. Chaum, Ed. Springer, 1984, pp. 51–67. [Online]. Available: https://doi.org/10.1007/978-1-4684-4730-9_5

[2] C. Cachin, "An information-theoretic model for steganography," *Information and Computation*, vol. 192, no. 1, pp. 41–56, 2004. [Online]. Available: https://doi.org/10.1007/3-540-49380-8_21

[3] N. Hopper, L. von Ahn, and J. Langford, "Provably secure steganography," *IEEE Transactions on Computers*, vol. 58, no. 5, pp. 662–676, 2009. [Online]. Available: https://doi.org/10.1109/TC.2008.199

[4] S. Berndt and M. Liśkiewicz, "On the gold standard for security of universal steganography," in *Advances in Cryptology – EUROCRYPT 2018*, J. B. Nielsen and V. Rijmen, Eds. Springer, 2018, pp. 29–60. [Online]. Available: https://doi.org/10.1007/978-3-319-78381-9_2

[5] T. Horel, S. Park, S. Richelson, and V. Vaikuntanathan, "How to subvert backdoored encryption: Security against adversaries that decrypt all ciphertexts," in *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), A. Blum, Ed., vol. 124. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018, pp. 42:1–42:20. [Online]. Available: https://doi.org/10.4230/LIPIcs.ITCS.2019.42

[6] T. Agrikola, G. Couteau, Y. Ishai, S. Jarecki, and A. Sahai, "On pseudorandom encodings," in *Theory of Cryptography*, R. Pass and K. Pietrzak, Eds. Springer, 2020, pp. 639–669. [Online]. Available: https://doi.org/10.1007/978-3-030-64381-2_23

[7] G. Kaptchuk, T. M. Jois, M. Green, and A. D. Rubin, "Meteor: Cryptographically secure steganography for realistic distributions," in *Proceedings of the 28nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. ACM, 2021, (accepted).

[8] A. Young and M. Yung, "Kleptography: Using cryptography against cryptography," in *Advances in Cryptology – EUROCRYPT '97*, W. Fumy, Ed. Springer, 1997, pp. 62–74. [Online]. Available: https://doi.org/10.1007/3-540-69053-0_6

[9] A. Russell, Q. Tang, M. Yung, and H.-S. Zhou, "Cliptography: Clipping the power of kleptographic attacks," in *Advances in Cryptology – ASIACRYPT 2016*, J. H. Cheon and T. Takagi, Eds. Springer, 2016, pp. 34–64. [Online]. Available: https://doi.org/10.1007/978-3-662-53890-6_2

[10] ——, "Generic semantic security against a kleptographic adversary," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. ACM, 2017, pp. 907—-922. [Online]. Available: https://doi.org/10.1145/3133956.3133993

[11] M. Bellare, K. G. Paterson, and P. Rogaway, "Security of symmetric encryption against mass surveillance," in *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds. Springer, 2014, pp. 1–19. [Online]. Available: https://doi.org/10.1007/978-3-662-44371-2_1

[12] M. Bellare, J. Jaeger, and D. Kane, "Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. ACM, 2015, pp. 1431—1440. [Online]. Available: https://doi.org/10.1145/2810103.2813681

[13] G. Ateniese, B. Magri, and D. Venturi, "Subversion-resilient signatures: Definitions, constructions and applications," *Theoretical Computer Science*, vol. 820, pp. 91–122, 2020. [Online]. Available: https://doi.org/10.1016/j.tcs.2020.03.021

[14] S. Berndt and M. Liśkiewicz, "Algorithm substitution attacks from a steganographic perspective," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. ACM, 2017, pp. 1649—1660. [Online]. Available: https://doi.org/10.1145/3133956.3133981

[15] R. Chen, X. Huang, and M. Yung, "Subvert KEM to Break DEM: Practical algorithm-substitution attacks on public-key encryption," in *Advances in Cryptology – ASIACRYPT 2020*, S. Moriai and H. Wang, Eds. Springer, 2020, pp. 98–128. [Online]. Available: https://doi.org/10.1007/978-3-030-64834-3_4

[16] L. M. Abbott, "Civil resistance in the Arab Spring: Triumphs and disasters," *International Affairs*, vol. 94, no. 1, pp. 212–213, 2016. [Online]. Available: https://doi.org/10.1093/ia/iix260

[17] P. N. Howard and M. M. Hussain, "The upheavals in Egypt and Tunisia: The role of digital media," *Journal of democracy*, vol. 22, no. 3, pp. 35–48, 2011. [Online]. Available: https://doi.org/10.1353/jod.2011.0041

[18] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *13th USENIX Security Symposium (USENIX Security 04)*. USENIX Association, 2004. [Online]. Available: https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router

[19] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh, "Stegotorus: A camouflage proxy for the Tor anonymity system," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. ACM, 2012, p. 109–120. [Online]. Available: https://doi.org/10.1145/2382196.2382211

[20] A. Dainotti, C. Squarcella, E. Aben, K. C. Claffy, M. Chiesa, M. Russo, and A. Pescapé, "Analysis of country-wide internet outages caused by censorship," *IEEE/ACM Transactions on Networking*, vol. 22, no. 06, pp. 1964–1977, 2014. [Online]. Available: https://doi.org/10.1109/TNET.2013.2291244

[21] D. Cole, "We Kill People Based on Metadata," May 10, 2014. [Online]. Available: https://www.nybooks.com/daily/2014/05/10/we-kill-people-based-metadata/

[22] G. J. Simmons, "The subliminal channel and digital signatures," in *Advances in Cryptology – EUROCRYPT '84*, ser. Lecture Notes in Computer Science, T. Beth, N. Cot, and I. Ingemarsson, Eds., vol. 209. Springer, 1985, pp. 364–378. [Online]. Available: https://doi.org/10.1007/3-540-39757-4_25

[23] ——, "Subliminal communication is easy using the DSA," in *Advances in Cryptology – EUROCRYPT '93*. Springer, 1994, pp. 218–232. [Online]. Available: https://doi.org/10.1007/3-540-48285-7_18

[24] M. Burmester, Y. G. Desmedt, T. Itoh, K. Sakurai, H. Shizuya, and M. Yung, "A progress report on subliminal-free channels," in *Information Hiding*, R. Anderson, Ed. Springer, 1996, pp. 157–168. [Online]. Available: https://doi.org/10.1007/3-540-61996-8_39

[25] J.-M. Bohli and R. Steinwandt, "On subliminal channels in deterministic signature schemes," in *Information Security and Cryptology – ICISC 2004*, C.-s. Park and S. Chee, Eds. Springer, 2005, pp. 182–194. [Online]. Available: https://doi.org/10.1007/11496618_14

[26] J.-M. Bohli, M. I. González Vasco, and R. Steinwandt, "A subliminal-free variant of ECDSA," in *Information Hiding*, J. L. Camenisch, C. S. Collberg, N. F. Johnson, and P. Sallee, Eds. Springer, 2007, pp. 375–387. [Online]. Available: https://doi.org/10.1007/978-3-540-74124-4_25

[27] M. Lepinksi, S. Micali, and a. shelat, "Collusion-free protocols," in *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '05. ACM, 2005, pp. 543—-552. [Online]. Available: https://doi.org/10.1145/1060590.1060671

[28] J. Alwen, J. Katz, Y. Lindell, G. Persiano, I. Visconti *et al.*, "Collusion-free multiparty computation in the mediated model," in *Advances in Cryptology - CRYPTO 2009*, S. Halevi, Ed. Springer, 2009, pp. 524–540. [Online]. Available: https://doi.org/10.1007/978-3-642-03356-8_31

[29] A. R. Hartl, R. Annessi, and T. Zseby, "A subliminal channel in EdDSA: Information leakage with high-speed signatures," in *Proceedings of the 2017 International Workshop on Managing Insider Security Threats*, ser. MIST '17. ACM, 2017, pp. 67—-78. [Online]. Available: https://doi.org/10.1145/3139923.3139925

[30] S. T. Ali, P. McCorry, P. H. Lee, and F. Hao, "ZombieCoin 2.0: Managing next-generation botnets using Bitcoin," *Int. J. Inf. Sec.*, vol. 17, no. 4, pp. 411–422, 2018. [Online]. Available: https://doi.org/10.1007/s10207-017-0379-8

[31] D. Frkat, R. Annessi, and T. Zseby, "ChainChannels: Private botnet communication over public blockchains," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1244–1252. [Online]. Available: https://doi.org/10.1109/Cybermatics_2018.2018.00219

[32] F. Gao, L. Zhu, K. Gai, C. Zhang, and S. Liu, "Achieving a covert channel over an open blockchain network," *IEEE Network*, vol. 34, no. 2, pp. 6–13, 2020. [Online]. Available: https://doi.org/10.1109/MNET.001.1900225

[33] Y. Zhang, H. Li, X. Li, and H. Zhu, "Provably secure and subliminal-free variant of Schnorr signature," in *Information and Communication Technology*, K. Mustofa, E. J. Neuhold, A. M. Tjoa, E. Weippl, and I. You, Eds. Springer, 2013, pp. 383–391. [Online]. Available: https://doi.org/10.1007/978-3-642-36818-9_42

[34] A. R. Hartl, R. Annessi, and T. Zseby, "Subliminal channels in high-speed signatures," *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 9, no. 1, pp. 30–53, 2018. [Online]. Available: https://doi.org/10.22667/JOWUA.2018.03.31.030

[35] S. Katzenbeisser and F. A. P. Petitcolas, "Defining security in steganographic systems," in *Security and Watermarking of Multimedia Contents IV*, E. J. Delp III and P. W. Wong, Eds., vol. 4675, International Society for Optics and Photonics. SPIE, 2002, pp. 50–56. [Online]. Available: https://doi.org/10.1117/12.465313

[36] L. von Ahn and N. Hopper, "Public-key steganography," in *Advances in Cryptology - EUROCRYPT 2004*, C. Cachin and J. L. Camenisch, Eds. Springer, 2004, pp. 323–341. [Online]. Available: https://doi.org/10.1007/978-3-540-24676-3_20

[37] N. Chandran, V. Goyal, R. Ostrovsky, and A. Sahai, "Covert multi-party computation," in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*. IEEE, 2007, pp. 238–248. [Online]. Available: https://doi.org/10.1109/FOCS.2007.61

[38] C. Cho, D. Dachman-Soled, and S. Jarecki, "Efficient concurrent covert computation of string equality and set intersection," in *Topics in Cryptology - CT-RSA 2016*, K. Sako, Ed. Springer, 2016, pp. 164–179. [Online]. Available: https://doi.org/10.1007/978-3-319-29485-8_10

[39] N. Fazio, A. R. Nicolosi, and I. M. Perera, "Broadcast steganography," in *Topics in Cryptology – CT-RSA 2014*, J. Benaloh, Ed. Springer, 2014, pp. 64–84. [Online]. Available: https://doi.org/10.1007/978-3-319-04852-9_4

[40] V. Goyal and A. Jain, "On the round complexity of covert computation," in *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, ser. STOC '10. ACM, 2010, pp. 191—200. [Online]. Available: https://doi.org/10.1145/1806689.1806717

[41] S. K. Jakobsen and C. Orlandi, "How to bootstrap anonymous communication," in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ser. ITCS '16. ACM, 2016, pp. 333—-344. [Online]. Available: https://doi.org/10.1145/2840728.2840743

[42] L. von Ahn, N. Hopper, and J. Langford, "Covert two-party computation," in *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '05. ACM, 2005, pp. 513—522. [Online]. Available: https://doi.org/10.1145/1060590.1060668

[43] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 2nd ed. CRC Press, Nov 2014.

[44] O. Goldreich, Ed., *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. ACM, 2019. [Online]. Available: https://doi.org/10.1145/3335741

[45] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Tech. Rep., 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[46] A. M. Antonopoulos, *Mastering Bitcoin: Programming the open blockchain*, 2nd ed. O'Reilly Media, Inc., Jul 2017. [Online]. Available: https://github.com/bitcoinbook/bitcoinbook

[47] J. Brendel, C. Cremers, D. Jackson, and M. Zhao, "The provable security of Ed25519: Theory and practice," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2021, pp. 1659–1676. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP40001.2021.00042

[48] *FIPS PUB 186-4: Digital Signature Standard (DSS)*, National Institute of Standards and Technology, Jul 2013. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.186-4

[49] M. Fersch, E. Kiltz, and B. Poettering, "On the one-per-message unforgeability of (EC)DSA and its variants," in *Theory of Cryptography*. Springer, 2017, pp. 519–534. [Online]. Available: https://doi.org/10.1007/978-3-319-70503-3_17

[50] J. Breitner and N. Heninger, "Biased Nonce Sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies," in *Financial Cryptography and Data Security*, I. Goldberg and T. Moore, Eds. Springer, 2019, pp. 3–20. [Online]. Available: https://doi.org/10.1007/978-3-030-32101-7_1

[51] D. R. L. Brown, *SEC 2: Recommended Elliptic Curve Domain Parameters*, Certicom Research, Jan 2010, version 2.0. [Online]. Available: https://www.secg.org/sec2-v2.pdf

[52] E. Fast, "Cryptography behind the top 100 cryptocurrencies," http://ethanfast.com/top-crypto.html, Feb 2021, accessed 06.08.2021, 17:03h.

[53] *NIST Special Publication 800-56A: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*, National Institute of Standards and Technology, Apr 2018, rev. 3. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-56Ar3

[54] D. R. L. Brown, *SEC 1: Elliptic Curve Cryptography*, Certicom Research, May 2009, version 2.0. [Online]. Available: https://www.secg.org/sec1-v2.pdf

[55] ——, "Generic groups, collision resistance, and ECDSA," *Des. Codes Cryptogr.*, vol. 35, no. 1, pp. 119–152, 2005. [Online]. Available: https://doi.org/10.1007/s10623-003-6154-z

[56] M. Fersch, E. Kiltz, and B. Poettering, "On the provable security of (EC)DSA signatures," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. ACM, 2016, pp. 1651—1662. [Online]. Available: https://doi.org/10.1145/2976749.2978413

[57] A. Bogdanov and A. Rosen, "Pseudorandom functions: Three decades later," in *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, Y. Lindell, Ed. Springer, 2017, pp. 79–158. [Online]. Available: https://doi.org/10.1007/978-3-319-57048-8_3

[58] S. Gueron, "Intel Advanced Encryption Standard (AES) New Instructions Set," Intel Corporation, May 2010, rev. 3.0, white paper. [Online]. Available: https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf

[59] A. Bhalla, "Top cryptocurrencies with their high transaction speeds," https://www.blockchain-council.org/cryptocurrency/top-cryptocurrencies-with-their-high-transaction-speeds/, Aug 2021, accessed 09.08.2021, 12:33h.

[60] V. Buterin, "Eth2 scaling for data," https://twitter.com/VitalikButerin/status/1277961594958471168, Jun 2020, accessed 09.08.2021, 12:45h.

[61] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering – Design Principles and Practical Applications*. Wiley, 2010, ch. 2.5, pp. 29–30.

[62] Blockchain, "Median confirmation time," https://www.blockchain.com/charts/median-confirmation-time, May 2021, accessed 04.05.2021, 19:40h.

[63] ——, "Average transactions per block," https://www.blockchain.com/charts/n-transactions-total, May 2021, accessed 04.05.2021, 19:39h.

[64] BitInfoCharts, "Bitcoin, XRP avg. transaction fee historical chart," https://bitinfocharts.com/en/comparison/transactionfees-btc-xrp.html, Jul 2021, accessed 09.08.2021, 15:32h.

[65] M. Singh, "WhatsApp is now delivering roughly 100 billion messages a day," https://techcrunch.com/2020/10/29/whatsapp-is-now-delivering-roughly-100-billion-messages-a-day/, Oct 2020, access 09.08.2021, 14:43h.

[66] J. Yin, X. Cui, C. Liu, Q. Liu, T. Cui, and Z. Wang, "CoinBot: A covert botnet in the cryptocurrency network," in *Information and Communications Security*, W. Meng, D. Gollmann, C. D. Jensen, and J. Zhou, Eds. Springer, 2020, pp. 107–125. [Online]. Available: https://doi.org/10.1007/978-3-030-61078-4_7

[67] A. I. Basuki and D. Rosiyadi, "Joint transaction-image steganography for high capacity covert communication," in *2019 International Conference on Computer, Control, Informatics and its Applications (IC3INA)*. IEEE, 2019, pp. 41–46. [Online]. Available: https://doi.org/10.1109/IC3INA48034.2019.8949606

[68] M. Abdulaziz, D. Çulha, and A. Yazici, "A decentralized application for secure messaging in a trustless environment," in *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*. IEEE, 2018, pp. 1–5. [Online]. Available: https://doi.org/10.1109/IBIGDELFT.2018.8625362

[69] L. Zhang, Z. Zhang, Z. Jin, Y. Su, and Z. Wang, "An approach of covert communication based on the Ethereum whisper protocol in blockchain," *International Journal of Intelligent Systems*, vol. 36, no. 2, pp. 962–996, 2021. [Online]. Available: https://doi.org/10.1002/int.22327

[70] S. Liu, Z. Fang, F. Gao, B. Koussainov, Z. Zhang, J. Liu, and L. Zhu, "Whispers on Ethereum: Blockchain-based covert data embedding schemes," in *Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, ser. BSCI '20. ACM, 2020, pp. 171—179. [Online]. Available: https://doi.org/10.1145/3384943.3409433

[71] J. Partala, "Provably secure covert communication on blockchain," *Cryptography*, vol. 2, no. 3, p. 18, 2018. [Online]. Available: https://doi.org/10.3390/cryptography2030018

[72] L. Zhang, Z. Zhang, W. Wang, R. Waqas, C. Zhao, S. Kim, and H. Chen, "A covert communication method using special Bitcoin addresses generated by Vanitygen," *Computers, Materials and Continua*, vol. 65, pp. 495–510, 2020. [Online]. Available: https://doi.org/10.32604/cmc.2020.011554

[73] J. Tian, G. Gou, C. Liu, Y. Chen, G. Xiong, and Z. Li, "DLchain: A covert channel over blockchain based on dynamic labels," in *Information and Communications Security*, J. Zhou, X. Luo, Q. Shen, and Z. Xu, Eds. Springer, 2020, pp. 814–830. [Online]. Available: https://doi.org/10.1007/978-3-030-41579-2_47

[74] S. S. M. Chow, A. Russell, Q. Tang, M. Yung, Y. Zhao, and H.-S. Zhou, "Let a non-barking watchdog bite: Cliptographic signatures with an offline watchdog," in *Public-Key Cryptography – PKC 2019*, D. Lin and K. Sako, Eds. Springer, 2019, pp. 221–251. [Online]. Available: https://doi.org/10.1007/978-3-030-17253-4_8

[75] Y. Wang, R. Chen, C. Liu, B. Wang, and Y. Wang, "Asymmetric subversion attacks on signature and identification schemes," *Personal and Ubiquitous Computing*, pp. 1–14, 2019. [Online]. Available: https://doi.org/10.1007/s00779-018-01193-x

APPENDIX

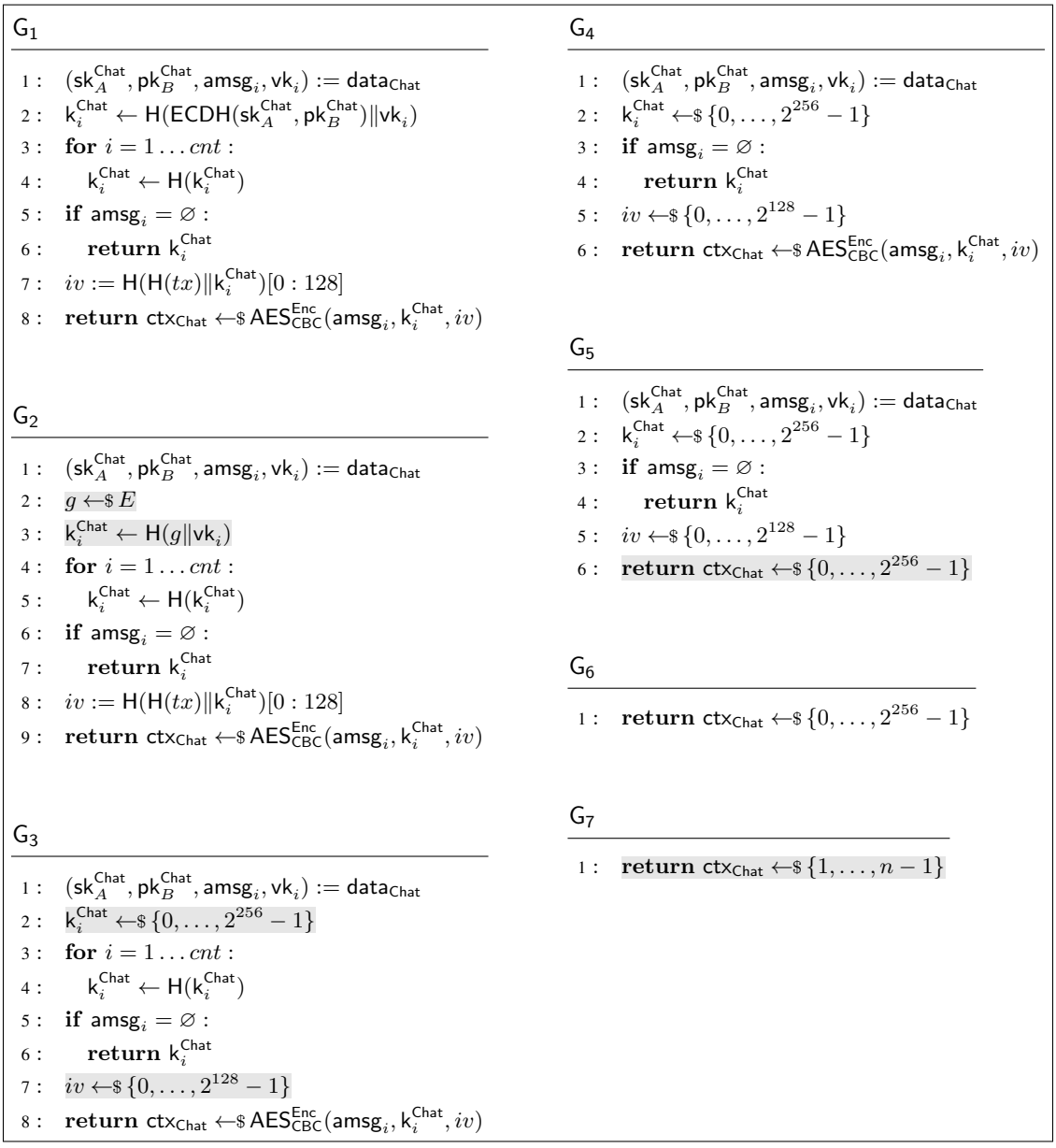

Figure 10. Games used in the proof of Lemma 1