"Act natural!": Exchanging Private Messages on Public Blockchains

Thore Tiemann
University of Lübeck
Lübeck, Germany
t.tiemann@uni-luebeck.de

Sebastian Berndt
University of Lübeck
Lübeck, Germany
s.berndt@uni-luebeck.de

Thomas Eisenbarth
University of Lübeck
Lübeck, Germany
thomas.eisenbarth@uni-luebeck.de

Maciej Liśkiewicz
University of Lübeck
Lübeck, Germany
liskiewi@tcs.uni-luebeck.de

***Abstract*—Messengers 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. Subliminal channels in cryptographic systems enable untraceable private communication in plain sight. In this context, bulletin boards in the form of 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 users generate hundreds of thousands of transactions per day that are individually signed and placed on the blockchain. Thus blockchains may serve as innocuous repository for publicly accessible cryptographic transactions where subliminal channels can be placed. In this paper, we propose a public-key subliminal channel using secret-recoverable splittable signature schemes on blockchains and prove that our construction is undetectable in the random oracle model under common cryptographic assumptions. Our approach is applicable to any secret-recoverable splittable signature scheme and introduces a constant overhead of a single signature per message. Such schemes are used by 98 of the top 100 cryptocurrencies. We also analyze the applicability of our approach to the Bitcoin, Monero, and RippleNet networks and present proof of concept implementations for Bitcoin and RippleNet.**

1. Introduction

The goal of steganography is to hide information in unsuspicious documents to achieve secret communication without revealing the presence of this 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 [63] and the investigation of steganography [11], [26], [44] and closely related topics such as kleptography [61], [74] and algorithm substitution attacks (ASAs) [8]–[10] 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, anonymity systems and steganography are related, but different. Using encryption, no eavesdropper that reads the ciphertext is able to obtain the *content* of the underlying plaintext. Using anonymity systems, Alice and Bob can hide their *identities* together with the 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.

When comparing anonymity systems to subliminal channels, anonymity systems usually achieve a higher bandwidth. However, users of anonymity systems like Tor [33] cannot create anonymity on their own but have to rely on the system's infrastructure that provides anonymity for its users. Even though an attacker is not able to identify the (content of the) messages sent over such a system, they can still detect that an anonymity system is used by analysing the metadata. This allows an attacker, if capable, to block the usage of the anonymity system [70]. Steganography on the other hand can be deployed on any existing benign channel that provides enough entropy. This means that no additional infrastructure is necessary and that the mere fact that steganography is used remains undetectable. 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 [30].

While subliminal channels have been formally introduced almost four decades ago, steganography has been mainly studied in the context of multimedia applications. Preventing subliminal communication in cryptographic systems is an important issue in cryptographic research, but there have not been as many works addressing this problem [3], [17], [18], [23], [49]. On the other hand, bulletin boards in the form of 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 current systems Tithonus [58] and MoneyMorph [53] use blockchains for steganography, but only implement embeddings in the *transaction scripts*. In constrast, the main focus of this paper concerns such channels in *digital*

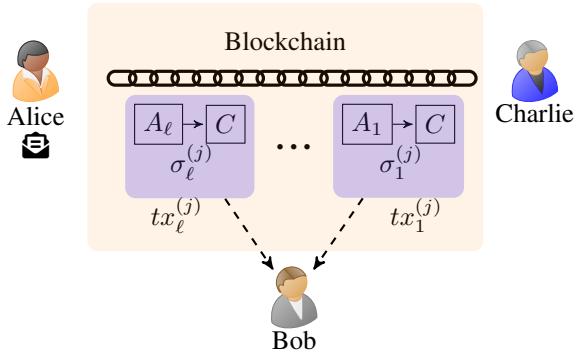

Figure 1. Subliminal chat from Alice to Bob: Alice signs and publishes transactions $tx_i^{(j)}$ for $i = 1, \ldots, \ell$ and $j = 1, 2$ on the blockchain. Bob analyzes the signatures and detects messages from Alice. Note that Charlie's identity is irrelevant for the chat and could be anyone, including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

signatures.

Recently, several other works propose also to implement covert channels in digital signature schemes, particularly in ECDSA and EdDSA which are commonly used in blockchain applications. It has been shown that both of them can be used for broadband subliminal communication [18], [43]. In [2], 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. [39] present an alternative construction, where the entire nonce can be used to get a broadband subliminal channel.

In [40], Gao et al. propose a kleptography-based digital signature algorithm to allow a subliminal communication 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.

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 We propose a public-key steganographic algorithm using secret-recoverable splittable signature schemes. Our approach works on *any* bulletin board featuring signed public messages. For simplicity, we focus on blockchains, as these are by far the most commonly used such boards. We describe and implement proof of concepts of our method for the Bitcoin and RippleNet networks, but our approach equally applies to 98 of the top 100 cryptocurrencies which rely on secret-recoverable splittable signature schemes like ECDSA or other variants of Schnorr signatures.

In our scenario, Alice and Bob communicate subliminally by embedding *private messages* in signatures of *public messages* on the bulletin board, i.e., transactions on a blockchain, which in turn transfer coins to a third non-suspicious party (cf. Fig. 1). They exchange private messages without having to meet a priori to agree upon a key or the third party. To this aim, additionally to the common asymmetric 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 the random nonce(s) during the generation of a signature. The secret-recoverability of the splittable signature schemes allows the receiver to recover the secret signing key. In subsequent signatures, the random nonce is replaced by pseudo-random ciphertexts. With the knowledge of the signing key, these ciphertexts can be extracted. 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 pseudo-random permutation. This is in contrast to most previous approaches which came without a formal security model or provable security analysis.

To the best of our knowledge, we present the first asymmetric stegosystem for covert communication on blockchain networks using signatures 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 efficient 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 [39].

To obtain such a system, we needed to overcome several challenges, including (a) the design of an efficient key distribution for bidirectional (subliminal) communication, (b) maintaining a provably secure solution with high-rate embedding, and (c) complying to the "one transaction per key" recommendation of UTXO blockchains.

Compared to previous solutions for steganography on bulletin boards, we (a) can make use of any bulletin board using splittable signature schemes, (b) have only a constant overhead of at most one signature per message, and (c) are able to reuse shared communication secrets that are independent from the signing keys used for the embedding. Finally, we (d) obtain an optimal embedding rate, as the complete entropy contained in a signature is used. Our global throughput is thus only limited by the bandwidth of the underlying bulletin board.

Nevertheless, our approach still shares some weaknesses with previous works, as (a) the receiver learns the secret signing key after completing the subliminal communication, (b) to achieve the best security guarantee, the receiver needs to scan the full bulletin board, and (c) the bandwidth of even very high-throughput blockchains is still very small compared to modern messengers.

The paper is organized as follows. Section 2 provides the needed preliminaries. Next, in Section 3, we formally define the security model and in Section 4 we provide the description and theoretical analysis of our method. Section 5 contains a discussion on aspects unique to each blockchain. In Section 6 we discuss in more detail the relevant, previous methods to hide messages in blockchain transactions.

2. Preliminaries

In our pseudocode, we write $x \| y$ to describe the concatenation of two strings 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 X and thus also write $x \leftarrow\!\!\$\ X$ for a random uniform assignment.

Provably Secure Steganography A complexity-theoretic model for symmetric steganography was proposed by Hopper, von Ahn, and Langford [44] and, independently, by Katzenbeisser and Petitcolas [46]. The asymmetric setting was first formalized by von Ahn and Hopper [67]. 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. [10], [27], [28], [35], [41], [45], [68]. 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. Steganography is often employed in very sensitive scenarios and we thus believe a provable guarantee to be of uttermost importance.

Blockchain Transactions

Bitcoin is the first cryptocurrency based on a public decentralized blockchain protocol. It was proposed in 2008 [54], 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.

Blockchains and ledgers like Bitcoin or Monero, rely on the *"unspent transaction output" (UTXO) model [32], [54]*. Transactions consist of a set of transaction inputs and outputs. Each output has a value and can only be spent once[1] and as a whole in a future transaction input. The value of an input defines the amount a user sends. The value of an output defines the amount a user receives. The difference[2] between the input sum and the output sum defines the transaction fee being payed to the miners. Transaction outputs always belong to a wallet. Wallets are not associated with an account balance but with a set of transaction outputs that were not spent, i.e., used as an input in another transaction, yet. However, a wallets balance can be computed by summing up all UTXOs belonging to this wallet. Therefore, UTXOs can be pictured as coins. They remain in owner's wallet until the owner decides to spend them in a transaction (and optionally receive change).

Blockchains and ledgers like Ethereum and RippleNet use the *account model* [24]. In this model, users posses accounts that are associated with an account balance, much like ordinary bank accounts. Receiving a transaction on an account increases the account's value and sending transactions from an account decreases its value. Transaction fees are defined explicitly as part of the transaction.

Digital signatures are a fundamental building block for all blockchains, due to their ability to guarantee the *authenticity* of transactions. A cryptocurrency wallet or account 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. In the UTXO model, it is used to sign the transaction outputs and those inputs that reference a corresponding wallet's UTXO. If multiple outputs of the same wallet are used as inputs in the same transaction, each input is signed individually. In the account model, sk is used to sign the transaction directly. So here we usually have a single signature per transaction and involved account. 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 some representation of the verification key. The address is used to receive transactions. Typically, a transaction also contains a *transaction script* which can be evaluated to obtain money sent in this transaction.

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 $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ (key generation, signing and verification) such that Vf is deterministic. and the algorithms have the following semantic: We say that SIG 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 $\sigma \in \mathrm{Supp}(\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}))$.

A scheme is called a *splittable signature scheme*, as defined in [69], if the Sign algorithm outputs signatures of the form $\sigma = (\sigma_R, \sigma_M)$. The two components are generated by two sub-algorithms respectively: $\mathsf{Sign}_R(k)$ takes some randomness k as input and outputs the randomness-binding component σ_R and $\mathsf{Sign}_M(\mathsf{sk}, \mathsf{msg}, k)$ takes the signing key sk, the message msg and the randomness k as input and outputs the message-binding component σ_M. A splittable signature scheme is called to be *secret-recoverable*, if a PPT algorithm can recover the signing key sk with high probability given a signature $\sigma = (\sigma_R, \sigma_M)$, the corresponding message msg, the verification key vk, *and* the randomness k used during the computation of σ.

Arguably, the *elliptic curve digital signature algorithm* (ECDSA) is the most widely used signature scheme. It is proven secure in the generic group model [38] and matches the definition of a splittable signature given in [69]. We denote the randomness-binding signature component by r and the message-binding component by s. The private signing key sk and the nonce k used during signing are the only unknowns in ECDSA. Therefore, it is easy to compute either value if the other becomes known. In particular, sk is easily computed as $\mathsf{sk} = (s \cdot k - h) r^{-1} \bmod n$ for some message hash h and group order n if k is revealed [19], which means that ECDSA is secret-recoverable.

Bitcoin recommends using a deterministic nonce generation algorithm instead of a pseudo random number generator (PRNG) to prevent nonce reuse or broken PRNGs,

1. This is enforced by the blockchain protocol: Blockchain validators reject transactions if they contain an input pointing to an output that is not part of the corresponding wallet's UTXO set and flag them as "double-spending".

2. The sum of inputs must be greater or equal the sum of outputs. Otherwise, the blockchain protocol rejects the transaction.

$\mathsf{NonceGenRFC6979}(\mathsf{H}(\mathsf{msg}), \mathsf{sk}, cnt)$

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

$\mathsf{SignECDSA}_{E,G,n}(\mathsf{sk}, \mathsf{msg})$

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

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

$\mathsf{SignMLSAG}_{E,G,n}(g, h, \pi, \mathsf{msg}, \{\mathsf{sk}_{\pi,j}\}, \{\mathsf{vk}_{i,j}\})$

1 : **for** $j \in \{1, 2, \ldots, h\}$:
2 : $\quad \widetilde{\mathsf{vk}}_j \leftarrow \mathsf{sk}_{\pi,j}\mathsf{H}(\mathsf{vk}_{\pi,j})$
3 : $\quad \alpha_j \leftarrow\!\!\$ \{1, 2, \ldots, n-1\}$
4 : $\quad$ **for** $i \in \{1, 2, \ldots, g\} \backslash \{\pi\}$:
5 : $\quad\quad r_{i,j} \leftarrow\!\!\$ \{1, 2, \ldots, n-1\}$
6 : $c_{\pi+1} := \mathsf{H}(\mathsf{msg}, ([\alpha_j G], [\alpha_j \mathsf{H}(\mathsf{vk}_{\pi,j})])_{1 \le j \le h})$
7 : **for** $i = \pi + 1, \pi + 2, \ldots, g, 1, 2, \ldots, \pi - 1$:
8 : $\quad$ **for** $j \in \{1, 2, \ldots, h\}$:
9 : $\quad\quad \zeta_{i,j} := r_{i,j} G + c_i \mathsf{vk}_{i,j}$
10 : $\quad\quad \gamma_{i,j} := r_{i,j} \mathsf{H}(\mathsf{vk}_{i,j}) + c_i \widetilde{\mathsf{vk}}_j$
11 : $\quad c_{i+1} := \mathsf{H}(\mathsf{msg}, (\zeta_{i,j}, \gamma_{i,j})_{1 \le j \le h})$
12 : **for** $j \in \{1, 2, \ldots, h\}$:
13 : $\quad r_{\pi,j} := \alpha_j - c_\pi \mathsf{sk}_{\pi,j} \pmod n$
14 : **return** $(c_1, r_{i,j})_{1 \le i \le g, 1 \le j \le h}, (\widetilde{\mathsf{vk}}_j)_{1 \le j \le h}$

Figure 3. Signing algorithm for MLSAG as described in [47].

which would allow for wallet key recovery through, e. g., lattice attacks [19]. RFC6979 defines one possible way for deterministic nonce generation. Figure 2 presents the ECDSA signature generation algorithm SignECDSA as it is implemented in libsecp256k1.

The *Elliptic Curve Schnorr Digital Signature Scheme* (EC-SDSA), a splittable signature scheme, was first described in [62]. As of today, several versions of this scheme are standardized. We denote the randomness-binding component of the signature with r and the message-binding component with s. If the nonce k used to compute (r, s) is known, the secret key can be computed as $\mathsf{sk} = (k - s)e^{-1} \bmod n$ for some hash e related to the signed message and group order n. Thus, the signatures are secret-recoverable. EdDSA is a popular variant of the Schnorr signature scheme and widely used in blockchains as well. For EdDSA the nonce is chosen deterministically by design [20]. The default curve is Curve25519 and the default hash function is SHA512. EdDSA with this configuration is called Ed25519. Most blockchains rely either on ECDSA or on Ed25519. Only two of the top 100 cryptocurrencies, namely IOTA and Arweave, do not use elliptic curve based signatures. The remaining 98 use ECDSA or a Schnorr signature variant [34].

The cryptocurrency Monero does not only use EdDSA for signing transactions, but additionally relies on *ring signatures* over an elliptic curve E, e. g., secp256k1, to hide the identity of the signer. The ring signature scheme used by Monero is the Multilayer Linkable Spontaneous Anonymous Group (MLSAG) signature scheme [47], [55]. In the following paragraph, we provide a detailed description of MLSAG and show that MLSAG is a secret-recoverable splittable signature. At the end of Section 4.4, we show an alternative way to embed messages into MLSAG signatures with a very high rate.

MLSAG generalizes the bLSAG signature scheme [47] to a set $\mathcal{K} = \{\mathsf{vk}_{i,j}\}$ of $g \cdot h$ keys ($i \in \{1, 2, \ldots, g\}$ and $j \in \{1, 2, \ldots, h\}$) where the signer knows the h private keys $\{\mathsf{sk}_{\pi,j}\}$ corresponding to the subset $\{\mathsf{vk}_{\pi,j}\}$ for some secret index $i = \pi$. The Sign algorithm for MLSAG is given in Fig. 3. It starts by computing key images for all public keys where the private key is known. Additionally, random values α_j and $r_{i,j}$ for all but the secret index $i = \pi$ are sampled. Then, $c_{\pi+1}$ is computed as the hash of the message, the curve points $\alpha_j \cdot G$ and the scalars $\alpha_j \mathsf{H}(\mathsf{vk}_{\pi,j})$. The remaining c_i are computed depending on the previous c_{i-1} in order. Here, c_i is the hash of the message, the curve points $\zeta_{i,j}$ and the scalars $\gamma_{i,j}$, which can be computed without the knowledge of corresponding private keys. Finally, the remaining values $r_{\pi,j}$ are computed. The resulting signature consist of the message-binding value c_1, the randomness-binding values $r_{i,j}$ and the key images $\widetilde{\mathsf{vk}}_j$. The scheme is secret-recoverable because, given the random values α_j, all secret keys can be computed as $\mathsf{sk}_j = (\alpha_j - r_{\pi,j}) c_\pi^{-1} \pmod n$, since all $r_{i,j}$ are given as part of the signature and all c_i values are either given in the signature (c_1) or can be computed. As π is not known, this step has to be done for all g possibilities and the resulting private key has to be checked against the corresponding public key to verify it. This is feasible and requires as much computational effort as verifying the signature.

ECDH The elliptic curve Diffie-Hellman (ECDH) protocol [22] allows Alice and Bob to exchange a secret symmetric key k through a public channel. The library *libsecp256k1* has support for ECDH. We use libsecp256k1 in Section 4.4 which is why we want to point out an important detail concerning the PoC: libsecp256k1 computes k by hashing the *compressed representation* of P. This 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 [22, Sec. B.4.1]. The compressed representation of P contains P_x as well as the *sign* of P_y. So hashing the compressed representation of (P_x, P_y) results in a different hash than hashing the compressed representation of $(P_x, -P_y)$.

Cryptographic Assumptions To show the security of our chat, we need three cryptographic assumptions 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 $1/2 + \epsilon$, if $\mathcal{D}$ makes at most q queries to their oracle. 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 at 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, Schnorr, and ring signatures already requires the random oracle model and also other assumptions which are not as standard as DDH or the hardness of AES [20], [21], [37], [38].

3. Our Model

Remember that in our setting, the goal is to transfer a private (hidden) message $\mathsf{text}^{\mathsf{Chat}}$ from Alice to Bob. To protect against an attacker monitoring the communication, 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 $\mathsf{sk}_A^{\mathsf{Chat}}$ and 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 wallets with addresses A_i and associated keys $(\mathsf{sk}_i, \mathsf{vk}_i)$ for the signature scheme. We do *not* assume that Bob has a wallet address himself, i.e., we do not assume that Bob is a participant in the blockchain network. We only require that Bob is able to look at transactions on the blockchain. In the following, we simply assume the existence of a third party, Charlie, with address A_C and keys $(\mathsf{sk}_C, \mathsf{pk}_C)$ that will receive the transactions sent by Alice. We stress that there is no need to trust Charlie, as he will not be able to detect the presence of the embedded communication. In a concrete application, Charlie might be a party which obtains many transactions, such as a charity. We discuss the choice of this third party and the resulting decisions more closely in Section 5. To start the communication, Alice and Bob exchange their public chat keys. This can be done via a Public Key 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 wallet 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 $\mathsf{text}^{\mathsf{Chat}}$ (not necessarily the content). In order to do so, we assume that they can participate in the blockchain network, i.e., they see all transactions sent over the network and can also send transactions themselves. Furthermore, we assume that Alice and Bob might be actively watched by $\mathcal{A}$. We thus assume that $\mathcal{A}$ has access to the public chat keys $\mathsf{pk}_A^{\mathsf{Chat}}$ and $\mathsf{pk}_B^{\mathsf{Chat}}$ of Alice and Bob. Hence, $\mathcal{A}$ can actively send information to Alice or Bob. In addition, 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 $\mathsf{text}^{\mathsf{Chat}}$, similar to a chosen-plaintext-attack. Furthermore, we assume that the attacker is able to observe the network traffic of both Alice and Bob and thus has access to the public addresses A_i and the verification keys vk_i of the wallets that Alice might use to send messages, the public address A_C of Charlie, and the corresponding verification key vk_C. As explained above, the choice of Charlie, the receiving party, is discussed in Section 5. As Bob only needs to observe transactions received by Charlie, this choice basically also determines Bob's behavior and is thus also discussed there.

In contrast to Moneymorph [53] and Tithonus [59], we will only embed information in the *signature* of arbitrary transactions. As all transactions in a blockchain contain such signatures, our theoretical model does not distinguish the different kind of transactions used by Alice, i.e., we assume in the following that the distribution of the transactions issued by Alice are indistinguishable from real transactions on the blockchain. As these distributions heavily rely on the choice of the concrete blockchain, we discuss these distributions and other important parameters such as money spent per transaction, frequency of transactions, etc. in more depth in Section 5. We stress here, that this is a strictly weaker assumption than those present in previous works, where a certain distribution (with sufficient min-entropy) on certain fields of a transaction is also required (see, e.g., [2], [65], [73]).

We give a formal security game involving this attacker below. We assume that $\mathsf{text}^{\mathsf{Chat}}$ is split into parts $\mathsf{text}_i^{\mathsf{Chat}}$ and each part $\mathsf{text}_i^{\mathsf{Chat}}$ is sent from a wallet with key-pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ for $i = 1, \ldots, \ell$. Moreover, we focus on the case that each $\mathsf{text}_i^{\mathsf{Chat}}$ can be embedded into a single signature.

3.1. Chat Scheme

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

- The key generation algorithm $\mathsf{KGenChat}$ is used to produce the chat key-pairs $(\mathsf{pk}^{\mathsf{Chat}}, \mathsf{sk}^{\mathsf{Chat}})$.
- The signing algorithm $\mathsf{SignChat}$ is given a secret signing key sk_i, a message msg_i, the public chat key $\mathsf{pk}_B^{\mathsf{Chat}}$ of Bob, the secret chat key $\mathsf{sk}_A^{\mathsf{Chat}}$ of Alice, and the hidden message $\mathsf{text}_i^{\mathsf{Chat}}$ and produces a signature σ_i for msg_i.
- The extraction algorithm $\mathsf{ExtChat}$ is used by Bob to extract a message $\mathsf{text}^{\mathsf{Chat}'}$ from several signatures σ_i. To do so, he is given public verification key vk_i, the public chat key $\mathsf{pk}_A^{\mathsf{Chat}}$ of Alice, and the secret chat key $\mathsf{sk}_B^{\mathsf{Chat}}$ of Bob.

Throughout this work, we always assume that Chat is correct, i.e., we require that $\mathsf{ExtChat}$ is able to reconstruct the original message $\mathsf{text}^{\mathsf{Chat}}$. Our proposed protocol will also guarantee perfect correctness. Note that some cryptocurrencies recommend to use an address for as few transactions as possible to achieve pseudonymity. Hence, sending many messages from the same address is detectable and thus suspicious behavior.

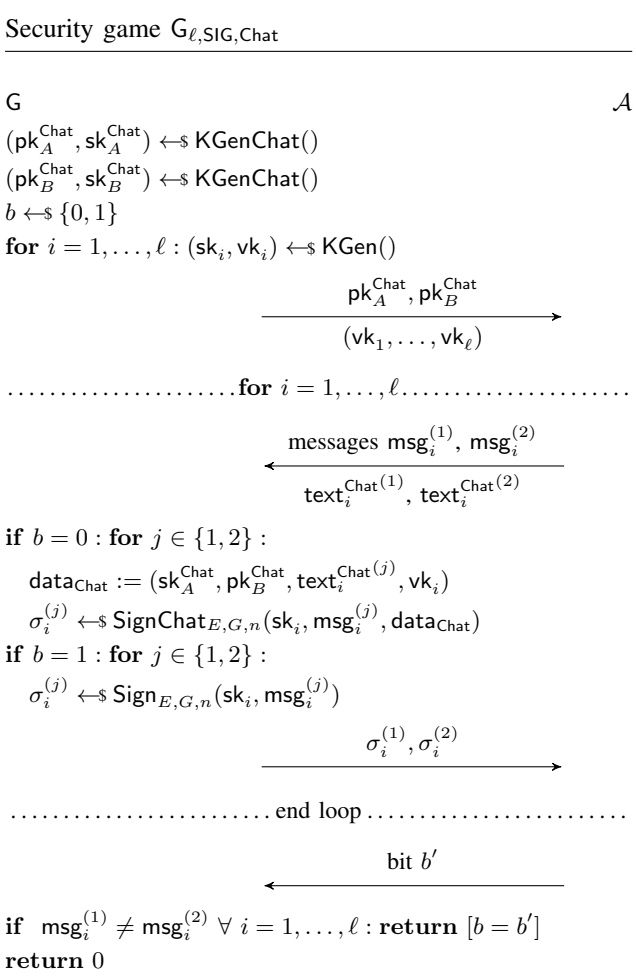

Figure 4. Security game to model undetectability. The attacker $\mathcal{A}$ chooses messages to sign and a message to embed. Their goal is to detect whether the signatures provided by challenger G contain an embedded message or not.

3.2. 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 and Alice controlling the users and sending signatures containing steganographic material. We stress that Alice does not need to to use all wallets at the same time, but rather might use them in an iterative fashion. 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 $\mathsf{text}^{\mathsf{Chat}}$ embedded into them. Note that in cryptocurrencies, each transaction is unique. To model this fact, we require that the messages queried by the attacker to be distinct.

Formally, this distinguishability game can be described as shown in Fig. 4. Here, the challenger G 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 G with two cover messages and an embeddable message and G signs both cover messages. Depending on the random bit, G optionally embeds the embeddable message into the cover messages during signing. Either way, the two signatures are returned to $\mathcal{A}$ who now has to determine whether the signatures contain the embeddable messages or not. Note that each signature key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ belongs to its own wallet, but all of the wallets are controlled by Alice, who has the chat 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 SignChat are indistinguishable, this is sufficient to argue for security in our model.

In the context of cryptocurrencies, we also want to protect against an embezzlement by Bob who follows the protocol but wants to steal Alice's coins. In this paper we call such a party *voracious Bob*. We stress here, that this does not mean that Bob will reveal the communication between Alice and him to the attacker. Initially, he knows $\mathsf{pk}_B^{\mathsf{Chat}}$, $\mathsf{sk}_B^{\mathsf{Chat}}$, vk_i, and $\mathsf{pk}_A^{\mathsf{Chat}}$, but neither $\mathsf{sk}_A^{\mathsf{Chat}}$ nor any sk_i. As Bob is aware that Alice wants to communicate with him, there is no need to hide this fact. His goal is rather to steal coins owned by Alice. A common way to leak information via secret-recoverable signatures applied in many of the previous works is to reveal the signing key sk_i in order to to recover the randomness/nonces k used in past transactions (which then include information about $\mathsf{text}^{\mathsf{Chat}}$). By learning Alice's signing key, Bob is enabled to impersonate Alice by signing messages in her name. In the context of cryptocurrencies, this allows Bob to obtain all of Alice's coins. To protect against such a voracious Bob, we thus need to guarantee that Alice has spent all of her coins before sk_i is revealed to Bob.

4. Subliminal Chat

We assume the following scenario: Alice sends a hidden message $\mathsf{text}^{\mathsf{Chat}}$ to Bob by transferring coins to Charlie. Alice has ℓ wallets, where wallet i has key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ 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 KGenChat. The scenario is shown in Fig. 1.

4.1. A Naive Approach

The basic idea of our chat protocol is to embed the hidden message $\mathsf{text}^{\mathsf{Chat}}$ in the nonce of ECDSA signatures. A similar approach was used in [39], but the authors focus on ECDSA and only allow unidirectional messaging. Our protocol allows for bidirectional communication and generalizes to secret-recoverable splittable signature schemes. To create the bidirectional subliminal channel, we propose a new way of leaking the secret signing key: Perform a non-interactive key exchange using the asymmetric key pairs generated by KGenChat, which results in a shared secret used to derive a nonce during the signing process to leak the secret signing key. We call this new approach NonceGenBasic. 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, only communication between the two parties using it is endangered.
- The subliminal channel is independent of Alice's and Bob's wallet addresses and keys. So, Alice and Bob do not need to know each others addresses in advance to communicate.

Obviously, using this naive approach is highly susceptible to resulting in a nonce reuse during signature generation which would undermine the security of all signature schemes presented 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.

4.2. 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. Note that we use a zero-round key exchange, and hence, there are no transactions necessary for session key exchange (for more discussion see Section 5). To send a message $\mathsf{text}^{\mathsf{Chat}}$ of b bytes to Bob, Alice splits $\mathsf{text}^{\mathsf{Chat}}$ into $\ell = \lceil \frac{b}{32} \rceil$ parts $\mathsf{text}_1^{\mathsf{Chat}}, \ldots, \mathsf{text}_\ell^{\mathsf{Chat}}$. We assume the length of $\mathsf{text}^{\mathsf{Chat}}$ to be a multiple of 32 bytes. If this should not be the case, we simply append null bytes to $\mathsf{text}^{\mathsf{Chat}}$. To embed $\mathsf{text}_i^{\mathsf{Chat}}$, Alice then creates transactions $tx_{A_i,C}^{(1)}$ and $tx_{A_i,C}^{(2)}$. 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 A_i with her signing key sk_i using the SignChat routine which makes use of NonceGenChat (cf. Fig. 5). She embeds her secret message part $\mathsf{text}_i^{\mathsf{Chat}}$ into the transaction $tx_{A_i,C}^{(1)}$ by passing it to the signing routine via $\mathsf{data}_{\mathsf{Chat}} = (\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}, \mathsf{text}_i^{\mathsf{Chat}}, \mathsf{vk}_i)$. Per call, NonceGenChat computes a shared secret $\mathsf{k}_i^{\mathsf{Chat}} = \mathsf{H}(\mathsf{ECDH}(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}) \| \mathsf{vk}_i)$ and encrypts the current message block $\mathsf{text}_i^{\mathsf{Chat}}$ with it. The resulting ciphertext is returned to be used as nonce by SignChat. We require the cipher to produce pseudo-random ciphertexts (which is stronger than indistinguishability of ciphertexts). 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 $(\tilde{t}, \tilde{\epsilon}, \tilde{q})$-pseudorandom permutation [16], AES-CBC produces (t, ϵ, q)-pseudorandom ciphertexts and is therefore suitable for our implementation (where $\tilde{t}$, $\tilde{\epsilon}$, and $\tilde{q}$ are close to t, ϵ, and q) [7]. More concretely, this means that ciphertexts produced by AES-CBC are (t, ϵ, q)-indistinguishable from uniform strings (upon random choice of the symmetric AES-key and adversarially chosen messages).

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

NonceGenChat is a deterministic algorithm. However, the nonce returned by NonceGenChat may lead to one of the message- or randomness-binding components to be zero. In this case, SignChat must 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. 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 (or l for the ring signatures) out of 2^{256} cases, i.e., it does not occur in practice.

When signing the second transaction $tx_{A_i,C}^{(2)}$, Alice now chooses $\mathsf{data}_{\mathsf{Chat}} = (\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}, \varnothing, \mathsf{vk}_i)$ which tells the nonce generation function to embed $\mathsf{k}_i^{\mathsf{Chat}}$ instead of a message. This way, the shared secret $\mathsf{k}_i^{\mathsf{Chat}}$ 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. Otherwise, a nonce reuse occurs which allows anyone monitoring the blockchain to take over her wallet.

After all transactions and signatures are generated, Alice publishes all transactions to the blockchain. In cases where Bob is assumed to be voracious, Alice has to take extra care when publishing $tx_{A_i,C}^{(2)}$ as it allows Bob to recover the 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 $tx_{A_i,C}^{(2)}$.

Bob on the other end follows the extraction algorithm depicted in Fig. 6 to receive $\mathsf{text}_i^{\mathsf{Chat}}$. He can compute $\mathsf{k}_i^{\mathsf{Chat}}$ from $\mathsf{sk}_B^{\mathsf{Chat}}$, $\mathsf{pk}_A^{\mathsf{Chat}}$, and vk_i. Bob searches the blockchain for a transaction $(tx_{A_i,C}^{(2)}, \sigma^{(2)})$, which in the case of ECDSA and Schnorr can be identified by r as $r^{(2)}$ is the x-coordinate of the point $\mathsf{k}_i^{\mathsf{Chat}} \cdot G$. In theory, an additional check for r being the x-coordinate of G multiplied with a hashed $\mathsf{k}_i^{\mathsf{Chat}}$ would be necessary. But since $\mathsf{k}_i^{\mathsf{Chat}}$ does not get hashed in practice (only with probability 2^{-255}), we discarded this test for simplicity. Because Bob knows the nonce used for signing $tx_{A_i,C}^{(2)}$, he can compute sk_i from either $(r^{(2)}, s^{(2)})$ or $(r^{(2)}, -s^{(2)})$. Decision for the correct sk_i is made by computing the corresponding verification key and comparing it with vk_i. This step is necessary in case the signature scheme in use requires normalized signatures (which is the case for libsecp256k1). For the ring signature scheme, Bob tries to reconstruct the secret signing key and verifies the signing key against the public key directly. Next, Bob searches the blockchain for the other transaction $(tx_{A_i,C}^{(1)}, \sigma^{(1)})$. Since Bob learned sk_i from $(tx_{A_i,C}^{(2)}, \sigma^{(2)})$, he can compute all the nonces used for signing $tx_{A_i,C}^{(1)}$ 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 $\mathsf{text}_i^{\mathsf{Chat}}$ 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 most cryptocurrencies or corresponding services assign a unique timestamp to all transactions. If Alice sends message part $\mathsf{text}_i^{\mathsf{Chat}}$ before $\mathsf{text}_{i+1}^{\mathsf{Chat}}$, Bob can determine the correct ordering of the parts by these timestamps.

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

1 : $(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}, \mathsf{text}^{\mathsf{Chat}}, \mathsf{vk}_i) := \mathsf{data}_{\mathsf{Chat}}$
2 : $\mathsf{k}_i^{\mathsf{Chat}} \leftarrow \mathsf{H}(\mathsf{ECDH}(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}) \| \mathsf{vk}_i)$
3 : **for** $j = 1 \dots cnt$:
4 : $\quad \mathsf{k}_i^{\mathsf{Chat}} \leftarrow \mathsf{H}(\mathsf{k}_i^{\mathsf{Chat}})$
5 : **if** $\mathsf{text}^{\mathsf{Chat}} = \varnothing$:
6 : $\quad$ **return** $\mathsf{k}_i^{\mathsf{Chat}}$
7 : $iv := \mathsf{H}(\mathsf{H}(\mathsf{msg}) \| \mathsf{k}^{\mathsf{Chat}})[0:128]$
8 : $\mathsf{ctx}_{\mathsf{Chat}} := \mathsf{AES}_{\mathsf{CBC}}^{\mathsf{Enc}}(\mathsf{text}^{\mathsf{Chat}}, \mathsf{k}_i^{\mathsf{Chat}}, iv)$
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 : $s := [k^{-1}(h + r \cdot \mathsf{sk}_i)] \bmod n$
7 : **if** $r = 0$ **or** $s = 0$:
8 : $\quad c \leftarrow c + 1$
9 : $\quad$ goto line 2
10 : **return** $(r, \min\{s, -s\})$

Figure 5. Nonce generation algorithm that enables the user to efficiently embed secret messages into the signatures in a provably undetectable way. The message msg to be signed is the transaction itself. The variable $\mathsf{data}_{\mathsf{Chat}}$ contains all additionally needed information for the embedding. Here, SignChat is implemented as a drop-in replacement for SignECDSA as an example.

We will now prove the security of our scheme for the case of ECDSA implemented on curve secp256k1. However, the proof is easily adjustable for other cryptographically secure elliptic curves and the other signature schemes discussed in Section 2.

Lemma 1. *Under the assumption that the $(t, \epsilon, 2)$-DDH assumption is true for secp256k1 and AES is a $(t, \epsilon, 1)$-pseudorandom permutation, the output of the algorithm $\mathsf{NonceGenChat}(\mathsf{H}(tx), \mathsf{data}_{\mathsf{Chat}}, cnt)$ and the uniform distribution on the set of integers $\{1, \dots, n-1\}$ are $(t, 3\epsilon + 11 \cdot 2^{-127}, 1)$-indistinguishable 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. 7. 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 g 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 $\mathsf{AES}_{\mathsf{CBC}}^{\mathsf{Enc}}$ is called with a secret key $\mathsf{k}_i^{\mathsf{Chat}}$ and a randomly chosen initialization vector iv, the generated ciphertext $\mathsf{ctx}_{\mathsf{Chat}}$

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

$Tx_C \leftarrow$ all (tx, σ) sending coins to A_C; $i \leftarrow 1$
foreach (tx, σ) **in** Tx_C :
$\mathsf{vk}_i, \mathsf{k}_i^{\mathsf{Chat}} :=$ $\mathsf{hasAmsg}(E, G, n, \mathsf{sk}_B^{\mathsf{Chat}}, \mathsf{pk}_A^{\mathsf{Chat}}, tx, \sigma)$
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** $r = x \bmod n$: **return** $\mathsf{vk}_i, \mathsf{k}_i^{\mathsf{Chat}}$
6 : **return** $0, 0$
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 : $h := \mathsf{H}(tx)$
2 : $(r, s) := \sigma$
3 : $\mathsf{sk}_i \leftarrow \left\lceil \frac{s \cdot \mathsf{k}_i^{\mathsf{Chat}} - h}{r} \right\rceil \bmod n$
4 : **if** $\mathsf{sk}_i \cdot G = \mathsf{vk}_i$: **return** sk_i
5 : $\mathsf{sk}_i \leftarrow \left\lceil \frac{(-s) \cdot \mathsf{k}_i^{\mathsf{Chat}} - h}{r} \right\rceil \bmod n$
6 : **if** $\mathsf{sk}_i \cdot G = \mathsf{vk}_i$: **return** sk_i
7 : **return** 0
if $\mathsf{sk}_i = 0$: continue
$Tx_{A,C} \leftarrow$ all $(tx_{A,C}, \sigma)$ where $tx_{A,C} \neq tx$
$\mathsf{text}_i^{\mathsf{Chat}} \leftarrow$ $\mathsf{recoverMsg}(n, \mathsf{k}_i^{\mathsf{Chat}}, \mathsf{sk}_i, Tx_{A,C})$
1 : **foreach** $(tx, (r, s))$ **in** $Tx_{A,C}$:
2 : $\quad h := \mathsf{H}(tx)$
3 : $\quad iv := \mathsf{H}(h \| \mathsf{k}_i^{\mathsf{Chat}})[0:128]$
4 : $\quad \mathsf{ctx}_{\mathsf{Chat}} \leftarrow \left\lceil \frac{\mathsf{sk}_i \cdot r + h}{s} \right\rceil \bmod n$
5 : $\quad tmp \leftarrow \mathsf{AES}_{\mathsf{CBC}}^{\mathsf{Dec}}(\mathsf{ctx}_{\mathsf{Chat}}, \mathsf{k}_i^{\mathsf{Chat}}, iv)$
6 : $\quad$ **if** tmp is meaningful : **return** tmp
7 : $\quad \mathsf{ctx}_{\mathsf{Chat}} \leftarrow \left\lceil \frac{\mathsf{sk}_i \cdot r + h}{-s} \right\rceil \bmod n$
8 : $\quad tmp \leftarrow \mathsf{AES}_{\mathsf{CBC}}^{\mathsf{Dec}}(\mathsf{ctx}_{\mathsf{Chat}}, \mathsf{k}_i^{\mathsf{Chat}}, iv)$
9 : $\quad$ **if** tmp is meaningful : **return** tmp
10 : **return** 0
$i \leftarrow i + 1$
reconstruct $\mathsf{text}^{\mathsf{Chat}}$ from $(\mathsf{text}_i^{\mathsf{Chat}})_{i=1,\dots,\ell}$

Figure 6. 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.

are $(t, 2\epsilon + 5 \cdot 2^{-128}, 1)$-pseudorandom and thus indistinguishable from a uniformly sampled string of length 256 [7, Theorem 17].

$\mathsf{G}_5 = \mathsf{G}_6$**:** The value of iv is not used in the algorithm anymore. Furthermore, the only part of $\mathsf{data}_{\mathsf{Chat}}$ still used is $\mathsf{text}_i^{\mathsf{Chat}}$. But $\mathsf{text}_i^{\mathsf{Chat}}$ is used only in the if-statement and in both cases ($\mathsf{text}_i^{\mathsf{Chat}} = \varnothing$ or $\mathsf{text}_i^{\mathsf{Chat}} \neq \varnothing$), 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[3].

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 $(t, 3\epsilon + 11 \cdot 2^{-127}, 1)$-indistinguishable from the uniform distribution on $\{0,1\}^{256}$. □

The above lemma implies that the nonce generated by our $\mathsf{NonceGenChat}$ algorithm 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 $\mathsf{NonceGenRFC6979}$ (cf. Fig. 2). Note that if every message is signed at most once (which is true for cryptocurrencies), this holds, as the distinguisher does not know the signing key sk_i.

Lemma 2. *The outputs of the algorithm* $\mathsf{NonceGenRFC6979}$ *with parameters* $(\mathsf{H}(\mathsf{msg}), d, cnt)$ *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. *Assuming 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* $(t, 4\epsilon + 11 \cdot 2^{-127})$*-undetectable for* $\ell \leq t$ *messages (with respect to ECDSA) in the random oracle model.*

Proof. Lemma 1 shows that under the given assumptions, random nonces and the generated nonces of $\mathsf{NonceGenChat}(\mathsf{H}(tx), \mathsf{data}_{\mathsf{Chat}}, cnt)$ are $(t, 3\epsilon + 11 \cdot 2^{-127}, 1)$-indistinguishable 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 $(t, 3\epsilon + 11 \cdot 2^{-127}, 1)$-indistinguishable from real nonces. The $(t, \epsilon, 2)$-security of ECDSA now implies that an attacker cannot extract the nonces from the signatures, as ECDSA is secret-recoverable. These nonces are thus not known by an attacker and the remaining part of $\mathsf{SignChat}$ are identical to ECDSA. Hence, Chat is $(t, 4\epsilon + 11 \cdot 2^{-127})$-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

3. Note that for secp256k1 $n = 2^{256} - 432420386565659656852420866394968145599$.

G_1

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

G_2

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

G_3

1 : $(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}, \mathsf{text}_i^{\mathsf{Chat}}, \mathsf{vk}_i) := \mathsf{data}_{\mathsf{Chat}}$
2 : $\mathsf{k}_i^{\mathsf{Chat}} \leftarrow\!\$\ \{0, \ldots, 2^{256} - 1\}$
3 : **for** $i = 1 \ldots cnt$: $\mathsf{k}_i^{\mathsf{Chat}} \leftarrow \mathsf{H}(\mathsf{k}_i^{\mathsf{Chat}})$
4 : **if** $\mathsf{text}_i^{\mathsf{Chat}} = \varnothing$: **return** $\mathsf{k}_i^{\mathsf{Chat}}$
5 : $iv \leftarrow\!\$\ \{0, \ldots, 2^{128} - 1\}$
6 : **return** $\mathsf{ctx}_{\mathsf{Chat}} \leftarrow\!\$\ \mathsf{AES}_{\mathsf{CBC}}^{\mathsf{Enc}}(\mathsf{text}_i^{\mathsf{Chat}}, \mathsf{k}_i^{\mathsf{Chat}}, iv)$

G_4

1 : $(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}, \mathsf{text}_i^{\mathsf{Chat}}, \mathsf{vk}_i) := \mathsf{data}_{\mathsf{Chat}}$
2 : $\mathsf{k}_i^{\mathsf{Chat}} \leftarrow\!\$\ \{0, \ldots, 2^{256} - 1\}$
3 : **if** $\mathsf{text}_i^{\mathsf{Chat}} = \varnothing$: **return** $\mathsf{k}_i^{\mathsf{Chat}}$
4 : $iv \leftarrow\!\$\ \{0, \ldots, 2^{128} - 1\}$
5 : **return** $\mathsf{ctx}_{\mathsf{Chat}} \leftarrow\!\$\ \mathsf{AES}_{\mathsf{CBC}}^{\mathsf{Enc}}(\mathsf{text}_i^{\mathsf{Chat}}, \mathsf{k}_i^{\mathsf{Chat}}, iv)$

G_5

1 : $(\mathsf{sk}_A^{\mathsf{Chat}}, \mathsf{pk}_B^{\mathsf{Chat}}, \mathsf{text}_i^{\mathsf{Chat}}, \mathsf{vk}_i) := \mathsf{data}_{\mathsf{Chat}}$
2 : **if** $\mathsf{text}_i^{\mathsf{Chat}} = \varnothing$:
3 : $\quad$ **return** $\mathsf{k}_i^{\mathsf{Chat}} \leftarrow\!\$\ \{0, \ldots, 2^{256} - 1\}$
4 : $iv \leftarrow\!\$\ \{0, \ldots, 2^{128} - 1\}$
5 : **return** $\mathsf{ctx}_{\mathsf{Chat}} \leftarrow\!\$\ \{0, \ldots, 2^{256} - 1\}$

G_6

1 : **return** $\mathsf{ctx}_{\mathsf{Chat}} \leftarrow\!\$\ \{0, \ldots, 2^{256} - 1\}$

G_7

1 : **return** $\mathsf{ctx}_{\mathsf{Chat}} \leftarrow\!\$\ \{1, \ldots, n-1\}$

Figure 7. Games used in the proof of Lemma 1.

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 blockchain node forwarding the transaction to him cannot be trusted. Message authenticity is guaranteed as only Alice owns the key sk_i used for signing the embedding transactions.

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 of the coins. 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 can receive messages without knowing the mailbox address (A_C in this case) by scanning all transactions in new blocks for messages from Alice. So changing the mailbox address while communicating is possible.

4.3. Considering multi-input transactions

So far we proposed and demonstrated a technique to securely send subliminal messages via general transactions. Our design, as well as other work like [39] require an overhead of one signature, which trivially translates to one transaction, to leak the signing key of the sender. Also, our technique requires the sender to issue two transactions from the same address. Since some currencies recommend to use addresses only once, our "chat behavior" might attract some attention. We can overcome both issues with a feature of cryptocurrencies based on the UTXO model like Bitcoin, Litecoin, Monero or Dash: *multi-input transactions*. UTXO model based cryptocurrencies allow (or even *require*) transactions to have multiple inputs (as shown later, these transactions are quite common). 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 RippleNet. 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.

4.4. PoC implementation

We implemented a proof of concept (PoC) of the chat client for the Bitcoin blockchain in Python available at https://github.com/UzL-ITS/act-natural. The program relies on the Python framework *bit*[4] to interact with the blockchain and manage keys. For key generation, signing, and verification, bit relies on *coincurve*[5] which is a Python wrapper for the highly optimized *libsecp256k1*. The library 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. We only have to implement AES and define the $\mathsf{data}_{\mathsf{Chat}}$ struct. Note that curve *secp256k1* is also used by Ethereum, RippleNet, Litecoin, EOS, and 80 other top 100 cryptocurrencies [34]. We implement AES following Intel's reference implementation for AES-NI as given in [42]. The code provides an efficient AES implementation while keeping the code base small and avoiding S-boxes or T-tables. We use the ECDH and SHA256 implementations of 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 accordingly.

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. Both transactions send bitcoins to an arbitrary letterbox 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. As the message exceeds 32 bytes, we need three inputs. The first two signatures contain the message and the third signature leaks sk_B to Alice. Both transactions transfer all remaining funds to a new wallet.

Additionally, we implemented a PoC client on the RippleNet network (we refer to RippleNet as Ripple for the remainder of the paper) in the same repository to show a client for account based cryptocurrencies as well. It makes use of the official library *xrpl-py* which in turn relies on the *ECPy* package for the cryptographic primitives. We extended ECPy to support ECDH and use *PyCryptodome*'s AES implementation to realize NonceGenChat. On the Ripple network, account creation and deletion cost US $10 and US $2, respectively. The transaction fee for regular transactions amounts to a fraction of a cent US. While each account address is directly tied to a *master keypair*, Ripple features *regular keypairs* that can be used for signature generation and verification. In case the key gets lost, the account holder can assign a new regular key. We exploit this feature to send arbitrarily many message from the same account without revealing the master key that proves account ownership.

4. https://github.com/ofek/bit

5. https://github.com/ofek/coincurve

TABLE 1. BITCOIN POC BENCHMARKS. ALL VALUES ARE AVERAGED OVER 1,000,000 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

4.5. Performance

We discuss the performance of our approach from two perspectives: First, we analyse the observable latency differences for common computations performed by the bulletin board participants. As we will show, no latency difference is observable which again supports the undetectability of our approach. Second, we discuss the capacity of modern blockchains and ledgers. Additionally, we propose another embedding for linkable ring signature that allows for an easy bandwidth extension.

Benchmarking the Bitcoin PoC shows that our approach is practical. The benchmarks we collected are depicted in Table 1. We measured the performance for the unmodified and the chat version of libsecp256k1. During signing and embedding, the usage of AES or additional hash operations do not add significant time penalties. The main timing difference of about $50\mu s$ between the libraries is caused by the new group operation due to the offline key exchange. This latency is only observable by the sender who is aware of using the steganography. The signature verification is not modified by our approach, so its latency is unchanged. These measurements show that our approach does not alter the observable time needed to verify a transaction or mine a new block as latency is only added in the signature generation step which is performed locally and offline by the sender.

Clearly, 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) 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 ledgers like EOS, or Ripple already offer a much higher capacity as they are capable of mining thousands of transactions per second [12]. This allows for a throughput of a least $1000\frac{\text{tx}}{\text{s}} \cdot 1\frac{\text{sig}}{\text{tx}} \cdot 32\frac{\text{byte}}{\text{sig}} = 32KB/s$. With english words containing on average 4.79 letters [56] and a typical text message containing 5.66 words [60], the average size of a text message is 27.11 bytes. Therefore, the global throughput today is >1,180 messages per second. These calculations show that the capacity of current blockchains and ledgers is not sufficient to replace modern messaging services like Signal or WhatsApp but can be used for highly confidential messages or to bootstrap, as, e.g., Tithonus [58]. We discuss the consequences of choosing a blockchain in the next section in more details.

For linkable ring signatures, which are used by Monero, an even simpler approach for embedding messages can be chosen that directly increases the bandwidth by $(g-1)\cdot h$: Generating a signature involves $(g-1)\cdot h$ random values $r_{i,j}$ $(i \neq \pi)$. To embed information in such a signature, one could compute a pseudorandom ciphertext (e. g., via AES-CBC) and split this ciphertext into these $(g-1)\cdot h$ random values. Due to the pseudorandom ciphertexts, these values look like random values, but Bob can use the exchanged shared key to recover the embedded information. The bandwidth per signature hereby increases from 32 bytes to $32\cdot(g-1)\cdot h$ bytes per signature.

5. Practical considerations for a real implementation

Before using our proof-of-concept (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) [36] as we do not require the keys to be kept secret (we even assume that $\mathsf{pk}_A^{\mathsf{Chat}}$ and $\mathsf{pk}_B^{\mathsf{Chat}}$ are known to the attacker). 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 (no matter the use case) is not to be known by an attacker, however, more sophisticated methods as, e. g., proposed in [48], [66] may be required.

Undetectability vs. performance Bob achieves highest undetectability by downloading the complete blockchain and searching it for new messages. While this might be feasible for slower blockchains like Bitcoin with a new blocks being mined every 8-15 minutes and containing 1500-2500 transactions each [14], [15], it quickly becomes infeasible for faster blockchains even if only new transactions have to be searched. On the other hand, Bob achieves optimal performance if Alice sends all transactions to a fixed mailbox address as Bob now only has to check for new transactions sent to this particular address. Of course, this comes at the cost of limited undetectability. By using a private signaling protocol as described in [51], Bob can achieve good performance while keeping a maximum of undetectability. This comes at the cost of additional infrastructure that is needed to implement the protocol. A good tradeoff relying solely on the blockchain can be achieved if the blockchain supports deterministic wallets [31], [71]. In this case, Alice can generate a sequence of wallets she owns, use them as mailbox addresses and share the seed for generating the sequence of corresponding wallet addresses with Bob. Sharing the seed can be done through our chat system at the cost of an initial higher cost or via out of band communication. Alice may also use deterministic wallets as her sending wallets rather than as the receiving wallets. This way, Bob can monitor the next sending wallet for publishing a transaction. As soon as Alice leaks the signing key of the first sending wallet to Bob, Bob can compute all future signing keys as well. This way, Alice can spare the transaction leaking a signing key for all future messages, effectively increasing the bandwidth. However, in this case

Bob has to be trustworthy as otherwise he could steal Alice's funds.

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 [13]. For applications with many users, the chosen blockchain should be capable of mining many transactions per second. For an example, EOS and Ripple are capable of handling thousands [12] and Ethereum 2.0 even promises 100.000 transactions per second [25]. While, this will still not be enough to handle services equivalent to WhatsApp which deals with about 1.2M messages per second [64], 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. Ripple, as an example, charges a fraction of a US cent per payment transaction [13].

Multiple transactions vs. multi-input transactions

We analyzed all Bitcoin, Monero, and Ripple transactions within the week February 07. - 13., 2022 for their number of inputs and outputs and several other properties. All details are given in Table 2 and Fig. 8 in the appendix. We found that Bitcoin and Monero transactions consist of 3.7368 and 2.1017 inputs per transaction on average while the median is one input for both blockchains. This metric does not apply to Ripple as it is an account based network instead of relying on the UTXO model. In general, it became obvious that transactions with more inputs are less likely to occur. A true chat client implementation should take this distribution for the chosen blockchain into account. This means that in practice, a balance between sending messages in multi-input transactions and multiple transactions should be chosen.

Dealing with a voracious receiver If Bob is assumed to be voracious, 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_i and therefore take control of 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_i to Bob if the blockchain uses the UTXO model. This is because Bob may learn the transaction before it is finally mined into a block. If Bob computes sk_i 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_i 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_i and all but little coins are transferred to a safe wallet. After the transaction is mined into a block, Alice may publish the transaction spending the remaining funds and leaking sk_i. Now Bob can read the message contained in the previous transaction but has no motivation to betray Alice. On the Ripple network, Alice does not have to worry about her coins. She can simply assign a new regular keypair to her account. As the assignment is done via a signed transaction, Alice can leak the old regular keypair in the very transaction that assigns the new keypair. As a transaction is fixed after three seconds and contains a sequence number, a voracious Bob can not intercept this.

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 [73]. Here, Bob basically behaves just like in the voracious case described above: After computing sk_i, Bob issues a new transaction with a higher transaction fee that double-spends the UTXOs Alice used for leaking sk_i. 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_i 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. Furthermore, an attacker might also look out for this kind of suspicious behaviour.

Avoiding traffic analysis While we assumed that the attacker has complete knowledge about the wallets used by Alice, there might be a situation where a traffic analysis can still be used to obtain unwanted information not covered by our model. In order to cover her tracks, Alice could make use of *mixing services* to hide the concrete wallets used by her. While the use of such services definitely increases the amount of work needed for the attacker, it does not completely prevent traffic analysis, as shown, e. g., in [52], [75]. Our analysis of the Bitcoin, Monero, and Ripple network shows that distributions of different transaction properties vary between the networks. In order to stay hidden, users of the chat client have to ensure that they generate transactions that *look normal*. Based on our observations we argue that it is easier to chat unnoticed on the Monero blockchain as it already hides some transaction metadata such as sender and receiver addresses or the amount of transferred funds [72]. Also, most transactions use the same type of ring signature and many transactions have two transaction inputs. While Ripple seems to be a good candidate when talking about transaction fees and throughput, a chat pattern might be easily detectable by traffic analysis. This is because transactions changing the regular keypair as well as transactions deleting an account are used rather seldom. Account deletions cost an extra fee of approx. 2 US dollar but changing the regular keypair is as expensive as any payment transaction.

TABLE 2. COMPARISON OF DIFFERENT STATISTICS FOR BITCOIN, MONERO, AND RIPPLE (FEB. 07 - 13, 2022).

	Bitcoin		Monero		Ripple	
	Average	Median	Average	Median	Average	Median
#Tx per addr & hour	0.007	0.006	0.003	0	0.241	0.0298
#Addr payed to per addr	3.052	1	hidden	hidden	6.724	1
#Inputs per tx	3.737	1	2.102	1	account	account
#Outputs per tx	3.332	2	2.408	2	account	account
Value per tx	21.382 BTC	0.0145 BTC	hidden	hidden	30950.375 XRP	100 XRP
Tx final latency[a]	56.881 min	52.1167 min	19.815 min	19.05 min	0.066 min	0.017 min

[a] Bitcoin: 6 blocks, Monero: 10 blocks, Ripple: next ledger

With ECDSA being used in 93.28% of all transactions and the knowledge that ECDSA is very vulnerable to bias attacks [4], we assumed that this transaction type would be used much more frequently. A problem of all three networks is the very low number of transactions per address and hour. The average Ripple account issues 0.241 transactions per hour. For Monero and Bitcoin, these values are even lower. For the UTXO-based blockchains Bitcoin and Monero, multiple addresses can be used to achieve higher throughput while keeping the number of transactions per address and hour low. In summary, users unafraid of metadata analysis may benefit from the high bandwidth and low transaction fees in the Ripple network. Users seeking for maximum anonymity, however, should rely on UTXO-model blockchains. From our point of view, achieving anonymity is easiest with Monero as the blockchain itself was designed with anonymity in mind. This comes at the cost of higher fees and lower bandwidth. With Monero, the bandwidth can be increased by exploiting the random values of the multi-signature scheme at the cost of even higher fees.

6. Related Work

Besides the works already mentioned in Section 1, there have been several other studies with the goal to build covert channels in blockchain systems. For example, Basuki et al. [6] introduce a covert channel encoding scheme based on smart contracts with a joint use of image steganography. Abdulaziz et al. [1] create a decentralized messaging application utilizing Whisper—the communication protocol of Ethereum—to send encrypted messages both securely and anonymously. In [76], similarly as in [1] and [50], 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 transferable to other blockchains. We circumvent this issue by embedding the messages in the output of cryptographic signatures which are widely used in blockchain systems.

Both MoneyMorph [53] and Tithonus [58] aim to establish a steganographic covert channel via blockchains. In contrast to our approach of embedding sensitive information in the signature present in each transaction, they embed information in the *transaction scripts*. As described in [53, Appendix A, Undetectability], the distribution of these transaction scripts might change very dynamically and both approaches thus need to constantly monitor the blockchain and adapt to possible changes. In contrast, *every* transaction contains a cryptographic signature. The distribution of those signatures are fully defined by the mathematics of the underlying signature scheme. Hence, we do not need a constant monitoring of the blockchain in our solution. Furthermore, in both MoneyMorph and Tithonus, the party receiving the transactions needs to take active part in the communication, whereas our approach allows to separate the recipient of the sensitive information from the receiver of the transaction. For the most common script type, pay-to-pubkey-hash, MoneyMorph can transfer 20 bytes of data per transaction. Note that these 20 bytes are bytes of the ciphertext and thus have a non-negligible overhead compared to the plaintext. In contrast, our approach transfers at least 16 bytes of *plaintext* per transaction and thus achieves a very similar bandwidth. The same is true for Tithonus. Finally, we note that MoneyMorph *always* burns money (except when using the very uncommon Pay2Multisig transaction) and thus has a limited number of at most 10^{12} messages on Bitcoin. Max-rate transactions introduced in [59] use a similar technique as MoneyMorph and Tithonus to implement a provably private and anonymous storage system. In contrast to our chat system, the storage system is limited to unidirectional communication as it proves the privacy and anonymity properties for the receiver only. Note that our approach is orthogonal to and directly compatible with both MoneyMorph and Tithonus, as both approaches only embed information in the transactions, while we embed into the signatures. We can thus easily combine the approaches to increase the bandwidth.

Ali et al. [2] 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 $\mathsf{PRF}_{\mathsf{k}^{\mathsf{Chat}}}((r, s)) = \mathsf{text}^{\mathsf{Chat}}$ for some pseudorandom function PRF. It is known that this approach is undetectable (see e. g. [10], [44]), but has a very limited rate logarithmic in the length of the nonce, i.e., at most 8 bits for curves of order nearly 256. Partala [57] 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. [77] improve the method of [57] 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 provably 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 [2], [73] or to identify transactions that embed

messages [65]. While Ali et al. [2] do not describe how to encode messages for the script, Yin et al. [73] 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. [65] 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 $\mathsf{text}^{\mathsf{Chat}}$. 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. [2] and Frkat et al. [39] who both embed messages by replacing the nonce with a symmetrically computed ciphertext of $\mathsf{text}^{\mathsf{Chat}}$. 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 there are in fact several parties constantly scanning the blockchain for such weak signatures (see, e.g., [19]). In addition to being detectable, an adversary can compute d and therefore impersonate the owner of the corresponding Bitcoin address. This is a major problem rendering this technique impractical.

To avoid the reuse of the nonce k, Frkat et al. [39] also propose a more involved method. The sender first symmetrically encrypts the messages $\mathsf{text}^{\mathsf{Chat}} = \mathsf{text}_1^{\mathsf{Chat}}, \ldots, \mathsf{text}_\ell^{\mathsf{Chat}}$ 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, since ECDSA is secret-recoverable, tries to recover the private signing key d by using $\mathsf{H}(\mathsf{k}^{\mathsf{Chat}})$ as described above. 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 $\mathsf{text}^{\mathsf{Chat}}$. 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. One might try to make the approach of Frkat et al. bi-directional by using two uni-directional channels. If all clients are provided with the same symmetric key, it is impossible for the command and control server to verify the identity of the sending client. Also, the private wallet key would always be revealed not only to an individual but a group of entities, which increases the chance of funds being stolen. If all clients use an individual key, the command and control server must send messages to each client individually which would increase the overhead linear to the number of clients and reduce the already limited bandwidth of the channel. In our chat scenario, this symmetric approach would thus require $O(n^2)$ keys if the system is used by n users and the provisioning of the symmetric keys remains an open problem. 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 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 agree on a key beforehand.

Another recent line of research, which is related to our work, deals with algorithm substitution attacks (ASA), and particularly with ASAs against digital signature schemes [5], [10], [29], [69]. 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 [69]). The resulting ASA would look similar to the attack given recently by Wang et al. [69], but has the advantage to embed arbitrary messages, like, e.g., sensitive data of users, and not just the signing key.

7. 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 PoC chat client for the Bitcoin blockchain and Ripple ledger.

Acknowledgements

We thank the anonymous reviewers for their supportive comments. A special thanks goes to Debajyoti Das for his detailed feedback and insightful tips on how to improve this work. This research was partially funded by the German Research Foundation (DFG) grant 456967092 (SecFShare).

References

[1] Mohamed Abdulaziz, Davut Çulha, and Ali Yazici. A decentralized application for secure messaging in a trustless environment. In *IBIGDELFT*, pages 1–5, 2018.

[2] Syed Taha Ali, Patrick McCorry, Peter Hyun-Jeen Lee, and Feng Hao. ZombieCoin 2.0: managing next-generation botnets using Bitcoin. *Int. J. Inf. Sec.*, 17(4):411–422, 2018.

[3] Joël Alwen, Jonathan Katz, Yehuda Lindell, Giuseppe Persiano, Abhi Shelat, and Ivan Visconti. Collusion-free multiparty computation in the mediated model. In *CRYPTO*, volume 5677 of *LNCS*, pages 524–540. Springer, 2009.

[4] Diego F. Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, and Yuval Yarom. LadderLeak: Breaking ECDSA with less than one bit of nonce leakage. In *CCS*, pages 225–242. ACM, 2020.

[5] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signatures: Definitions, constructions and applications. *Theor. Comput. Sci.*, 820:91–122, 2020.

[6] Akbari Indra Basuki and Didi Rosiyadi. Joint transaction-image steganography for high capacity covert communication. In *IC3INA*, pages 41–46. IEEE, 2019.

[7] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *FOCS*, pages 394–403. IEEE, 1997.

[8] Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In *CCS*, pages 1431–1440. ACM, 2015.

[9] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In *CRYPTO (1)*, volume 8616 of *LNCS*, pages 1–19. Springer, 2014.

[10] Sebastian Berndt and Maciej Liśkiewicz. Algorithm substitution attacks from a steganographic perspective. In *CCS*, pages 1649–1660. ACM, 2017.

[11] Sebastian Berndt and Maciej Liśkiewicz. On the gold standard for security of universal steganography. In *EUROCRYPT*, volume 10820 of *LNCS*, pages 29–60. Springer, 2018.

[12] Anshika 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.

[13] 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.

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

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

[16] Andrej Bogdanov and Alon Rosen. Pseudorandom functions: Three decades later. In *Tutorials on the Foundations of Cryptography*, pages 79–158. Springer, 2017.

[17] Jens-Matthias Bohli and Rainer Steinwandt. On subliminal channels in deterministic signature schemes. In *ICISC*, volume 3506 of *LNCS*, pages 182–194. Springer, 2004.

[18] Jens-Matthias Bohli, María Isabel González Vasco, and Rainer Steinwandt. A subliminal-free variant of ECDSA. In *IH*, volume 4437 of *LNCS*, pages 375–387. Springer, 2006.

[19] Joachim Breitner and Nadia Heninger. Biased Nonce Sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies. In *FC*, volume 11598 of *LNCS*, pages 3–20. Springer, 2019.

[20] Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. The provable security of Ed25519: Theory and practice. In *SP*, pages 1659–1676. IEEE, 2021.

[21] Daniel R. L. Brown. Generic groups, collision resistance, and ECDSA. *Des. Codes Cryptogr.*, 35(1):119–152, 2005.

[22] Daniel R. L. Brown. *SEC 1: Elliptic Curve Cryptography*. Certicom Research, May 2009. version 2.0.

[23] Mike Burmester, Yvo Desmedt, Toshiya Itoh, Kouichi Sakurai, Hiroki Shizuya, and Moti Yung. A progress report on subliminal-free channels. In *Information Hiding*, volume 1174 of *LNCS*, pages 157–168. Springer, 1996.

[24] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform, 2014.

[25] Vitalik Buterin. ETH2 scaling for data. https://twitter.com/VitalikButerin/status/1277961594958471168, Jun 2020. accessed 09.08.2021, 12:45h.

[26] Christian Cachin. An information-theoretic model for steganography. *Inf. Comput.*, 192(1):41–56, 2004.

[27] Nishanth Chandran, Vipul Goyal, Rafail Ostrovsky, and Amit Sahai. Covert multi-party computation. In *FOCS*, pages 238–248. IEEE, 2007.

[28] Chongwon Cho, Dana Dachman-Soled, and Stanislaw Jarecki. Efficient concurrent covert computation of string equality and set intersection. In *CT-RSA*, volume 9610 of *LNCS*, pages 164–179. Springer, 2016.

[29] Sherman S. M. Chow, Alexander Russell, Qiang Tang, Moti Yung, Yongjun Zhao, and Hong-Sheng Zhou. Let a non-barking watchdog bite: Cliptographic signatures with an offline watchdog. In *PKC (1)*, volume 11442 of *LNCS*, pages 221–251. Springer, 2019.

[30] David Cole. We kill people based on metadata, May 10, 2014.

[31] Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. In *CCS*, pages 651–668. ACM, 2019.

[32] Sergi Delgado-Segura, Cristina Pérez-Solà, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomartí. Analysis of the bitcoin UTXO set. In *Financial Cryptography Workshops*, volume 10958 of *Lecture Notes in Computer Science*, pages 78–91. Springer, 2018.

[33] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320. USENIX, 2004.

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

[35] Nelly Fazio, Antonio Nicolosi, and Irippuge Milinda Perera. Broadcast steganography. In *CT-RSA*, volume 8366 of *LNCS*, pages 64–84. Springer, 2014.

[36] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering - Design Principles and Practical Applications*. Wiley, 2010.

[37] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the provable security of (EC)DSA signatures. In *CCS*, pages 1651–1662. ACM, 2016.

[38] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the one-per-message unforgeability of (EC)DSA and its variants. In *TCC (2)*, volume 10678 of *LNCS*, pages 519–534. Springer, 2017.

[39] Davor Frkat, Robert Annessi, and Tanja Zseby. ChainChannels: Private botnet communication over public blockchains. In *iThings/GreenCom/CPSCom/SmartData*, pages 1244–1252. IEEE, 2018.

[40] Feng Gao, Liehuang Zhu, Keke Gai, Can Zhang, and Sheng Liu. Achieving a covert channel over an open blockchain network. *IEEE Netw.*, 34(2):6–13, 2020.

[41] Vipul Goyal and Abhishek Jain. On the round complexity of covert computation. In *STOC*, pages 191–200. ACM, 2010.

[42] Shay Gueron. Intel Advanced Encryption Standard (AES) New Instructions Set, May 2010. rev. 3.0, white paper.

[43] Alexander Hartl, Robert Annessi, and Tanja Zseby. A subliminal channel in EdDSA: Information leakage with high-speed signatures. In *MIST@CCS*, pages 67–78. ACM, 2017.

[44] Nicholas J. Hopper, Luis von Ahn, and John Langford. Provably secure steganography. *IEEE Trans. Computers*, 58(5):662–676, 2009.

[45] Sune K. Jakobsen and Claudio Orlandi. How to bootstrap anonymous communication. In *ITCS*, pages 333–344. ACM, 2016.

[46] Stefan Katzenbeisser and Fabien A. P. Petitcolas. Defining security in steganographic systems. In *Security and Watermarking of Multimedia Contents*, volume 4675 of *SPIE Proceedings*, pages 50–56. SPIE, 2002.

[47] Koe, Kurt M. Alonso, and Sarang Noether. *Zero to Monero: A technical guide to a private digital currency; for beginners, amateurs, and experts*. Monero, 2020. v2.0.0.

[48] David Lazar and Nickolai Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *OSDI*, pages 571–586. USENIX Association, 2016.

[49] Matt Lepinski, Silvio Micali, and Abhi Shelat. Collusion-free protocols. In *STOC*, pages 543–552. ACM, 2005.

[50] Shaoyuan Liu, Zhi Fang, Feng Gao, Bakh Khoussainov, Zijian Zhang, Jiamou Liu, and Liehuang Zhu. Whispers on Ethereum: Blockchain-based covert data embedding schemes. In *BSCI*, pages 171–179. ACM, 2020.

[51] Varun Madathil, Alessandra Scafuro, István András Seres, Omer Shlomovits, and Denis Varlakov. Private signaling. In *USENIX Security Symposium*, pages 3309–3326. USENIX, 2022.

[52] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of Bitcoins: characterizing payments among men with no names. *Commun. ACM*, 59(4):86–93, 2016.

[53] Mohsen Minaei, Pedro Moreno-Sanchez, and Aniket Kate. MoneyMorph: Censorship resistant rendezvous using permissionless cryptocurrencies. *Proc. Priv. Enhancing Technol.*, 2020(3):404–424, 2020.

[54] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[55] Shen Noether and Adam Mackenzie. Ring confidential transactions. *Ledger*, 1:1–18, 2016.

[56] Peter Norvig. English letter frequency counts: Mayzner revisited or ETAOIN SRHLDCU. http://norvig.com/mayzner.html. accessed: 2021-11-18 21:25.

[57] Juha Partala. Provably secure covert communication on blockchain. *Cryptogr.*, 2(3):18, 2018.

[58] Ruben Recabarren and Bogdan Carbunar. Tithonus: A Bitcoin based censorship resilient system. *Proc. PETs*, 2019(1):68–86, 2019.

[59] Ruben Recabarren and Bogdan Carbunar. Toward uncensorable, anonymous and private access over Satoshi blockchains. *Proc. Priv. Enhancing Technol.*, 2022(1):207–226, 2022.

[60] Avi Rosenfeld, Sigal Sina, David Sarne, Or Avidov, and Sarit Kraus. A study of WhatsApp usage patterns and prediction models without message content. *CoRR*, abs/1802.03393, 2018.

[61] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic semantic security against a kleptographic adversary. In *CCS*, pages 907–922. ACM, 2017.

[62] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, volume 435 of *LNCS*, pages 239–252. Springer, 1989.

[63] Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In *CRYPTO*, pages 51–67. Plenum Press, New York, 1983.

[64] Manish 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.

[65] Jing Tian, Gaopeng Gou, Chang Liu, Yige Chen, Gang Xiong, and Zhen Li. DLchain: A covert channel over blockchain based on dynamic labels. In *ICICS*, volume 11999 of *LNCS*, pages 814–830. Springer, 2019.

[66] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *SOSP*, pages 137–152. ACM, 2015.

[67] Luis von Ahn and Nicholas J. Hopper. Public-key steganography. In *EUROCRYPT*, volume 3027 of *LNCS*, pages 323–341. Springer, 2004.

[68] Luis von Ahn, Nicholas J. Hopper, and John Langford. Covert two-party computation. In *STOC*, pages 513–522. ACM, 2005.

[69] Yi Wang, Rongmao Chen, Chi Liu, Baosheng Wang, and Yongjun Wang. Asymmetric subversion attacks on signature and identification schemes. *Pers. Ubiquit. Comput.*, 26(3):849–862, 2022.

[70] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is blocking Tor. In *FOCI*. USENIX, 2012.

[71] Pieter Wuille. Bip-32: Hierarchical deterministic wallets. https://github.com/bitcoin/bips/blob/64aba767e2d422b5d471509acc340750432613ae/bip-0032.mediawiki, 2022. accessed 23.08.2022, 16:06h.

[72] Claire Ye, Chinedu Ojukwu, Anthony Hsu, and Ruiqi Hu. Alt-coin traceability. *IACR Cryptol. ePrint Arch.*, page 593, 2020.

[73] Jie Yin, Xiang Cui, Chaoge Liu, Qixu Liu, Tao Cui, and Zhi Wang. CoinBot: A covert botnet in the cryptocurrency network. In *ICICS*, volume 12282 of *LNCS*, pages 107–125. Springer, 2020.

[74] Adam L. Young and Moti Yung. Kleptography: Using cryptography against cryptography. In *EUROCRYPT*, volume 1233 of *LNCS*, pages 62–74. Springer, 1997.

[75] Haaroon Yousaf, George Kappos, and Sarah Meiklejohn. Tracing transactions across cryptocurrency ledgers. In *USENIX Security Symposium*, pages 837–850. USENIX, 2019.

[76] Lejun Zhang, Zhijie Zhang, Zilong Jin, Yansen Su, and Zhuzhu Wang. An approach of covert communication based on the Ethereum whisper protocol in blockchain. *Int. J. Intell. Syst.*, 36(2):962–996, 2021.

[77] Lejun Zhang, Zhijie Zhang, Weizheng Wang, Rasheed Waqas, Chunhui Zhao, Seokhoon Kim, and Huiling Chen. A covert communication method using special Bitcoin addresses generated by Vanitygen. *Computers, Materials and Continua*, 65(1):597–616, 2020.

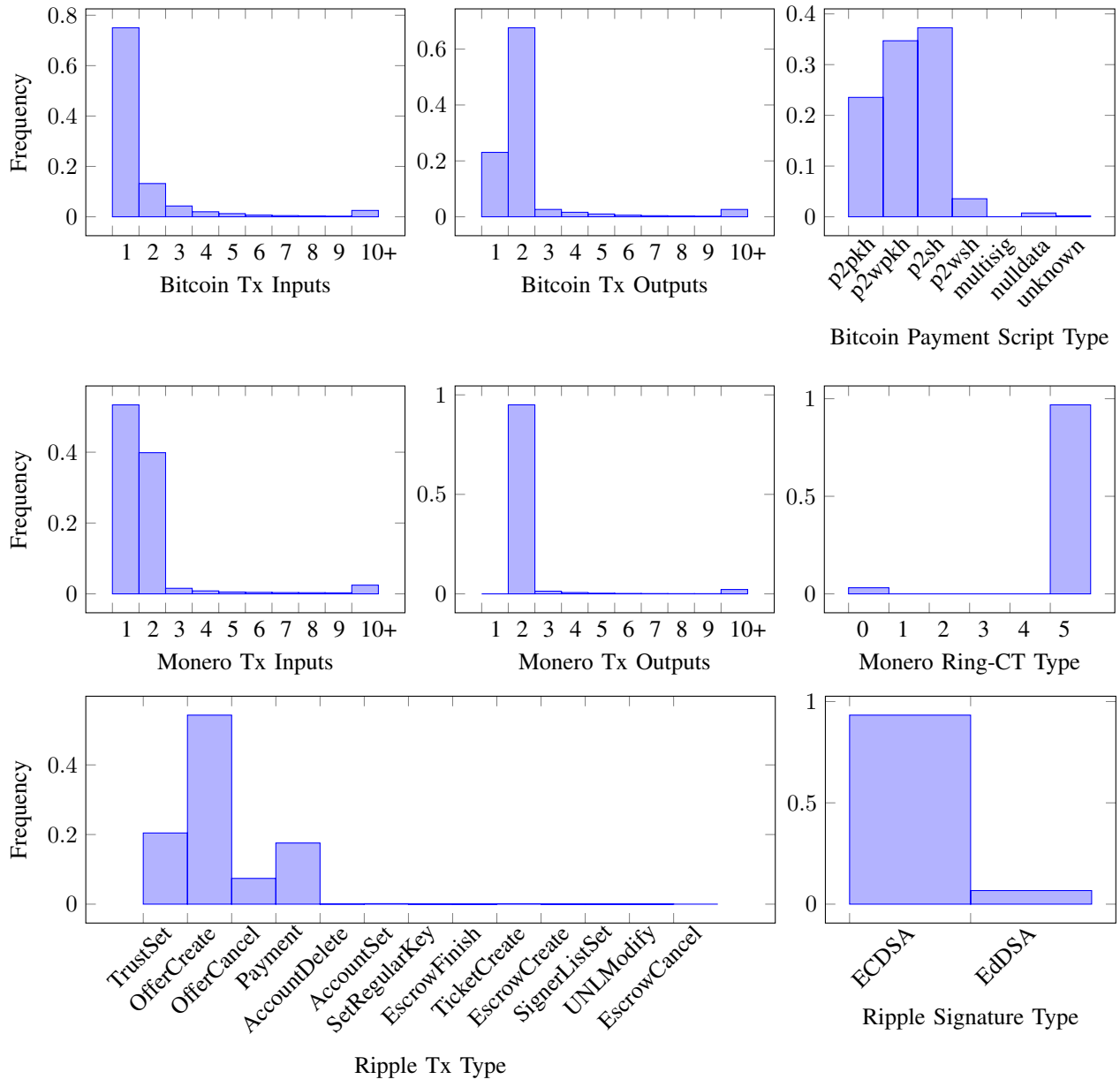

Figure 8. Distribution plots of different properties of Bitcoin, Monero and Ripple transactions (Feb. 07 - 13, 2022).