Uncontrolled Randomness in Blockchains: Covert Bulletin Board for Illicit Activities

Nasser Alsalami and Bingsheng Zhang

Lancaster University, UK
{n.alsalami,b.zhang2}@lancaster.ac.uk

Abstract. The blockchain technology represents a new paradigm to realize persistent distributed ledgers globally. While the blockchain technology is promising in a great number of fields, it can be abused to covertly store and disseminate potentially harmful digital content. Consequently, using blockchains as uncensored decentralized networks for arbitrary data distribution poses a serious regulatory issue. In this work, we show the severity of the problem by demonstrating a new technique that can be exploited to use the blockchain as a covert bulletin board to secretly store and distribute objectionable content. More specifically, all major blockchain systems use randomized cryptographic primitives, such as digital signatures and non-interactive zero-knowledge proofs, and we illustrate how the uncontrolled randomness in such primitives can be maliciously manipulated to enable covert communication and hidden persistent storage. We also demonstrate how the same technique can be extended to launch subversion attacks on the wallets of most top-ranked cryptocurrencies, such as Bitcoin, Ethereum, Monero, etc. To clarify the potential risk of uncontrolled randomness, we design, implement and evaluate our technique against the widely-used ECDSA signature scheme, the CryptoNote's ring signature scheme, and Monero's ring confidential transactions. Note that the significance of the demonstrated attacks stems from their undetectability, their adverse effect on the future of decentralized blockchains, and their serious repercussions on users' privacy and crypto funds. Finally, besides presenting the attacks, we provide a discussion of current countermeasures and suggest some countermeasures to mitigate the threat of such attacks.

Keywords: Blockchain, Steganography, Covert Broadcast Channels, Content Insertion, Wallet Subversion

Table of Contents

1 Introduction

The blockchain technology has pioneered a new paradigm to realize large-scale immutable, persistent, and append-only distributed ledgers. Nowadays, blockchain-powered systems have become largely ubiquitous across various sectors including technology, academia, medicine, economics, finance, etc. While the blockchain technology is promising in a great number of application scenarios, it can also be used to anonymously store and disseminate potentially harmful digital content. A recent study [1] has shown that 1.4% of all Bitcoin transactions contain non-financial data, some of which contain objectionable content, e.g. links to child pornography. The absence of a central authority makes it hard to censor the content posted on decentralized blockchains. With the increasing amount of illicit content posted to the blockchains on daily basis, using blockchains as uncensored networks for arbitrary data storage and distribution has become a serious regulatory issue. Subsequently, several techniques have been discussed to either filter unwanted content before it is added to the ledger [2] or remove content from the blockchain [3, 4].

However, all of the proposed countermeasures can only be effective if the malicious content attached to the transactions can be detected. The situation gets worse when the attackers can hide data into normal transactions and use blockchain platforms for covert communications. Naively, one can encrypt the malicious content and attach its ciphertext to a transaction, but it is noticeable to the public that there is suspicious data attached. In 2018, Partala [5] showed a proof-of-concept steganography technique that allows an adversary to covertly embed one bit into a standard blockchain transaction without being distinguished from an innocuous transaction.

In this work, we further advance this line of research by demonstrating an effective steganographic method that offers high throughput and can be launched against any blockchain platforms that use randomized, i.e. probabilistic, cryptographic primitives, such as digital signatures and non-interactive zero-knowledge proofs. The main observation is that all randomized cryptographic algorithms need to consume random coins somewhere along the execution, and these random coins are not audited or certified publicly. By intentionally manipulating the random coin supplied to a randomized algorithm, an attacker is able to embed arbitrary information into the outputs of the algorithm. The output that contain steganographic data is computationally indistinguishable from normal output.

Besides using the demonstrated attack for covert channels and hidden storage, the same attack is applicable in another scenario. The attacker(s) may try to subvert, or mis-implement, cryptocurrency wallets and re-distribute them to unsuspecting users. The subverted wallets can then surreptitiously leak the victim's secret, such as the signing key, via standard transactions. Importantly, the transactions generated by the subverted wallet are computationally indistinguishable from normal transactions for any black-box observer. We emphasize that this attack scenario is very realistic and represents a serious vulnerability. Currently the focus of research regarding blockchain subversion vulnerabilities is

mainly on the trusted parameter setup process, such as common reference string (CRS) generation [6, 7], while software subversion vulnerabilities in blockchain cryptocurrency applications has not been extensively studied. The plausibility of algorithm-substitution attacks against cryptocurrency can be attributed to the following three reasons.

Firstly, cryptocurrencies have very complex cryptographic primitives and structures. This complexity requires highly sophisticated mathematical and cryptographic expertise to effectively review the source code and implementation. As a result, cryptographic design and implementation mistakes can be found in the products of many world-leading IT companies. For instance, as recently shown in [8], Tencent's QQ browser uses textbook RSA algorithm with no padding, which is well-known to be insecure as it is a deterministic encryption scheme. Likewise, it is shown in [9] that over 1/3 of the smart contracts , which are open-source, contain at least one bug, and some of them are maliciously embedded and can be triggered later by the attackers in a similar manner to the infamous Ethereum DAO hack [10] ($55 million USD).

Secondly, the highly centralized development may cause bias and introduce intentional and unintentional flaws which may not be spotted by code reviewers. Although many cryptocurrencies are marketed as open-source decentralized platforms, in many cases, the majority of the source code is contributed by a single developer or a small group of developers. Studies have found that the development of many blockchain applications is highly centralized in reality. For example, 30% of the source files in Bitcoin are written by a single author, and 7% of the code is written by the same author [11]. Similarly, 20% of the source code in Ethereum is attributed to the same author [11].

Thirdly, common end users lack the ability and the means to check the conformity of an executable wallet with its reference source code. In fact, in some platforms, such as iOS, users can not directly access the binary files without jailbreaking their devices, which paradoxically is not advisable and may render a device unsafe to run a cryptocurrency wallet. Besides, it is uncommon for users to compile the source code of any application by themselves; instead, they usually relay on downloading readily prepared executable applications. The difficulty to examine the implementation of a cryptocurrency wallet is even more pertinent to hardware wallets, such as the various *Swiss-Army-Knife* hardware wallets [12]. These hardware wallets are typically manufactured in an outsourced loosely-controlled environment, and it is practically impossible to audit the integrity of their implementation through the standard functionality 'correctness' test by observing input/output pairs in a black-box manner.

Our contributions. The primary objective of this work is to draw attention to the potential threat of abusing uncontrolled randomness in blockchain algorithms. To the best of our knowledge, this work is the first in literature that discusses such a widely spread vulnerability in the blockchain context. More specifically, we summarize our contributions as follows:

– **Novel blockchain steganographic technique.** We propose a steganographic technique that greatly increases the throughput of the state-of-the-

art blockchain steganographic attack that affect many cryptocurrencies. We present our general attack against the widely-used CryptoNote framework, and as a demonstration, we design, implement and evaluate the attack on Monero and Bytecoin currencies.

- **Covert broadcast channels.** As an immediate application, we show blockchain platforms can be exploited to act as covert broadcast channels. Once deployed, this would be the world's first practical covert broadcast channel. The existence of such a channel will be untraceable, unlinkable, and even unobservable. Such broadcast channels could be disastrous if used by outlaws, e.g. terrorists.
- **Persistent storage.** With the proposed steganographic technique, anyone can use the blockchain as a cheap hidden persistent storage along with his/her daily transactions. For instance, this can be used for uncensorable cyberlockers. At the time of submission, persistently storaging 1G data on Bytecoin and using its P2P network as CDN only costs less than 3 USD. In theory, data storage is just a communication channel between the current user and the user himself/herself in the future. Nevertheless, there is a subtle difference between hidden storage and covert channels, that is how long the channel (data) would exist. Some countermeasures are effective against persistent storage but not covert channels.
- **Wallet subversion attacks**. For the first time, we point out that there is a troubling high risk of massive coin stealing among all of the current cryptocurrency wallets by demonstrating efficient and effective subversion attacks within the realm of *Kleptography* and *Algorithm Substitution* Attacks. This attack possesses the following properties:
  - *Passive attack.* The attacker does not need to interact directly with the victim's wallet, i.e. subverted wallet. The communication channel between the subverted wallets and the attacker is simply through the transactions posted on the blockchain.
  - *Undetectability.* The transactions generated by compromised wallets are computationally indistinguishable from the normally generated transactions. Therefore, no online/offline *watchdog* can detect the subversion.
  - *Interoperability.* The subverted wallets transact seamlessly with normal wallets; i.e. they can send to and receive from other wallets regardless whether other wallets are subverted or not.
  - *Subtlety.* Although we present our attack in the black-box setting, when optimized, the difference between a subverted wallet source code, e.g. Bytecoin wallet, and the original code is only about ten lines of code in two functions. This subtlety makes it difficult even for technology-savvy users to review and detect the subversion even if the subverted wallet is open source.

  We have implemented our subversion attacks against the ECDSA signature scheme, and the ring signature used in the CryptoNote framework which is implemented by many cryptocurrencies, such as Bytecoin. Because ECDSA and ring signature are widely used among cryptocurrencies, this work has direct impact on 18 of the top 25 cryptocurrencies in terms of market capitalization [13] (as of the time of writting) as depicted in Table 1.

Table 1. Cryptocurrencies and Digital Signature Schemes (currencies checked with either the ECDSA signature or the Ring signature are potentially susceptible to the our wallet subversion attacks.)

Cryptocurrencies' Signatures					
#	**Cryptocurrency**	**ECDSA**	**EdDSA**	**Ring Signature**	**Note**
1	Bitcoin	✓			
2	Ethereum	✓			
3	Ripple	✓	✓		
4	Bitcoin Cash	✓			
5	Litecoin	✓			
6	Cardanos		✓		
7	Stellar		✓		
8	Zcash		✓		
9	IOTA				Winternitz
10	Monero			✓	
11	Dash	✓			
12	NEM		✓		
13	Ethereum Classic	✓			
14	Komodo	✓			
15	Verge	✓			
16	Lisk		✓		
17	Dogecoin	✓			
18	Decred	✓	✓		
19	Nano		✓		
20	Wanchain	✓		✓	
21	Bytecoin			✓	
22	Siacoin		✓		
23	Bitcoin Diamond	✓			
24	BitShares	✓			
25	Waves		✓		

1.1 Paper Roadmap

The rest of this document is organized as follows: Sec. 2 provides background and definitions, and explains some preliminary concepts. In Sec. 3, we present our generic steganographic attack against CryptoNote-based cryptocurrencies, illustrate the effectiveness of the attack, and discuss its security proof. After that, we demonstrate our implementation of the generic steganographic attack in Bytecoin and Monero in Sec. 4. In Sec. 5 and Sec. 6, we explore three different scenarios in which our generic attack could be applied, besides Sec. 6 presents two more subversion attacks on ECDSA-signature wallets. We also study the current countermeasures and suggest some mitigations in Sec. 7, and provide the related work in Sec. 8. Finally, Sec. 9 restates the main objectives and findings, concludes the document, and explains potential future work.

2 Preliminaries

Below we describe the necessary notations to used in this document, and provide description of some preliminary concepts that are related to this work.

2.1 Notations

We use the following notations throughout this paper. The notation $[n]$ stands for the set $\{1, 2, \ldots, n\}$. For a randomized algorithm $A()$, we write $y = A(x; r)$ to denote the unique output of A on input x and randomness r, and write $y \leftarrow A(x)$ to denote the process of picking randomness r uniformly at random and setting $y = A(x; r)$. We use $s \xleftarrow{\$} S$ to denote sampling an element s uniformly at random from a set S. We use $\lambda \in \mathbb{N}$ as the security parameter. Let $\mathsf{poly}(\cdot)$ denote a polynomially-bounded function and $\mathsf{negl}(\cdot)$ denote a negligible function. Unless specified in the context, we use $\mathsf{hash}_p : \{0,1\}^* \mapsto \mathbb{Z}_p$ and $\mathsf{hash}_g : \{0,1\}^* \mapsto \mathbb{G}$ as two collision resistant hash functions that map an arbitrary length string to a group element in $\mathbb{Z}_p$ and $\mathbb{G}$, respectively. However, the actual group parameters of $\mathbb{Z}_p$ and $\mathbb{G}$ may vary depending on the context. $m_{[a:b]}$ stands for the truncation that contains from the a-th bit to the b-th bit of m.

2.2 Blockchain

The term *blockchain* encompasses a broader range of distributed ledger technologies initiated by the Bitcoin [14]. There are two types of blockchain; permissioned (private) and permissionless (public). In this work, we mainly focus on permissionless blockchain. Typically, a permissionless blockchain uses a *Proof-of-X* mechanism, such as Proof-of-Work (PoW) and Proof-of-Stake (PoS), to randomly nominate a node who will propose the next block. The valid transactions contained in a block need to be signed by the owner(s) of the corresponding consumed coins. Most blockchain systems use randomized signature algorithms, which makes them vulnerable to our attacks.

2.3 (Ring) Signature schemes

For notation simplicity, we unify the syntax of signature schemes and ring signature schemes. A (ring) signature scheme consists of a tuple of algorithms $\mathcal{S} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ as follows:

- $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$ is the setup algorithm that takes as input the security parameter 1^λ, and it outputs a system parameter param. The rest of the algorithms implicitly take param as an input.
- $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$ is the key generation algorithm that takes as input the setup parameter param, and it outputs a public key and secret key pair $(\mathrm{PK}, \mathrm{SK})$.

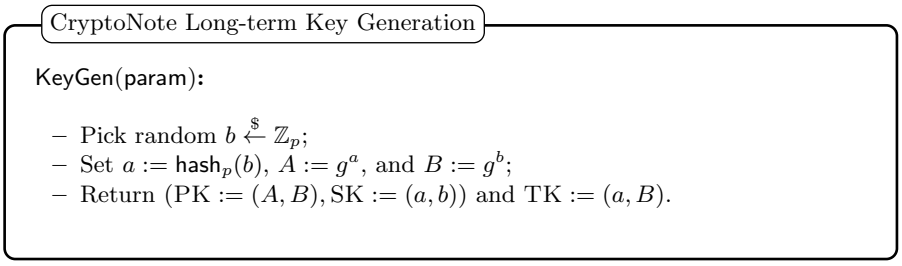

Fig. 1. CryptoNote Long-term Key Generation Algorithm.

- $\sigma \leftarrow \mathsf{Sign}(\mathcal{P}, \mathrm{SK}, \ell, m)$ is the signing algorithm that takes as input a set of public keys $\mathcal{P} := \{\mathrm{PK}_1, \ldots, \mathrm{PK}_n\}$, the secret key SK, the index ℓ such that SK is the corresponding secret key of PK_ℓ, and the message m, and it outputs the signature σ. (For a standard signature scheme, we have $|\mathcal{P}| = 1$ and $\ell = 1$.)
- $b \leftarrow \mathsf{Verify}(\mathcal{P}, m, \sigma)$ is the verification algorithm that takes as input a set of public keys $\mathcal{P}$, the message m and the signature σ, and it outputs $b := 1$ if only if the signature is valid.

Unforgeability. In a blockchain application, a signature scheme needs to achieve existential unforgeability under an adaptive chosen-message attack (EUF-CMA). Whereas, there are various unforgeability definitions for ring signature schemes; in this work, we adopt the most commonly used unforgeability against fixed-ring attacks and unify it with EUF-CMA. We refer interested readers to [15] for more ring signature security definition variants and their differences.

Definition 1. *We say a (ring) signature scheme* $\mathcal{S} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is EUF-CMA if for any PPT adversary* $\mathcal{A}$*, any integer* $\lambda \in \mathbb{N}$*, any* $n = \mathsf{poly}(\lambda)$*, any* $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$*, any* $\{(\mathrm{PK}_i, \mathrm{SK}_i)\}_{i=1}^n$ *output by* $\mathsf{KeyGen}(\mathsf{param})$*, we have:*

$$\Pr\left[\begin{array}{l}(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot,\cdot)}(\{\mathrm{PK}_i\}_{i=1}^n) : \\ \mathsf{Verify}(\{\mathrm{PK}_i\}_{i=1}^n, m^*, \sigma^*) = 1 \ \wedge \ m^* \notin \mathcal{Q}\end{array}\right] = \mathsf{negl}(\lambda)$$

where $\mathcal{O}(s, m) := \mathsf{Sign}(\{\mathrm{PK}_i\}_{i=1}^n, \mathrm{SK}_s, s, m)$ *be the signing oracle, and* $\mathcal{Q} := \{m_1, \ldots, m_q\}$ *is the set of queries to the signing oracle* $\mathcal{O}(\cdot, \cdot)$*.*

2.4 Brief description of CryptoNote

CryptoNote is a protocol proposed by Nicolas van Saberhagen [16], and it has been implemented in many emerging cryptocurrencies, such as Bytecoin [17], CryptoNoteCoin [18], Fantomcoin [19], etc. Compared to Bitcoin-like cryptocurrencies, CryptoNote offers two main features: (i) *stealth address via non-interactive key exchange* and (ii) *set anonymity via (linkable) ring signatures*. More specifically, the user's private key consists of $a, b \in \mathbb{Z}_p$, and the corresponding public key (A, B) consists of $A := g^a$ and $B := g^b$. Note that in a standard CryptoNote

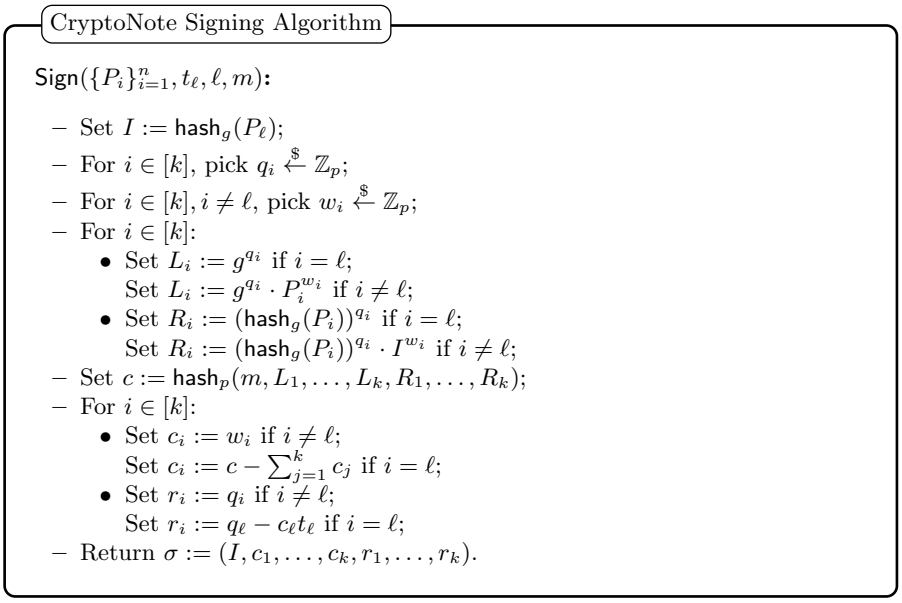

Fig. 2. CryptoNote Signing Algorithm.

implementation, a is usually defined as $\mathsf{hash}_p(b)$; therefore, b is the actual secret key. To transfer funds to a recipient, the payer needs to generate a transaction public key $R := g^r$ and compute the corresponding one-time address $T := (g^{\mathsf{hash}_p(A^r)} \cdot B)$. The recipient is then able to compute the corresponding one-time private key as $t := (\mathsf{hash}_p(R^a) + b)$. By the property of Diffie-Hellman exchange, we have $A^r = R^a$. With regards to the one-time ring signature schemes, it is transformed from the OR-composition of Schnorr's identification Sigma protocols. There exists a LNK algorithm that can link two signatures together if they are produced by the same signing key. By design, the one-time signature key can only be used once, and it can be detected if the same key is used to sign two transactions, which prevents double spending. More specifically, let $T := g^t$ be the one-time public key, and define $I := (\mathsf{hash}_g(T))^t$ as a "key image" as part of the signature. The ring signatures signed by the same secret key would have identical key image; therefore, double spending can be defeated efficiently by simply checking if the key image has already been used.

Let $\mathcal{P} := \{P_i\}_{i=1}^n$ be a set of public keys, and the signer knows the secret key t_ℓ such that $P_\ell = g^{t_\ell}$, $\ell \in [n]$. Denote $I := \mathsf{hash}_g(P_\ell)$ as the key image. The param is defined as the parameters of the ED25519 twisted Edwards curve. For completeness, we provide the key generation and signing algorithms in Fig. 1 and Fig. 2, respectively. Fig. 1 shows a third key called the *tracking key* TK that can be given to a third party to track all transactions destined to the owner of this key without revealing their secret key SK.

2.5 Brief description of Monero (Version 0.12.0.0)

Monero [20] is one of the most successful CryptoNote-based cryptocurrencies, and its source code is available on GitHub [21]. Although the original Monero was based on the CryptoNote protocol, its transaction signature has evolved beyond this protocol[1]. As mentioned in [23], CryptoNote suffers from a shortcoming where amounts in transactions are not hidden. To address this issue, *Ring Confidential Transaction* (RingCT) [23] has been developed and deployed in Monero. It combines (linkable) ring signature and Pedersen commitment schemes [24], and also adopts Multilayered Linkable Spontaneous Anonymous Group Signature (MLSAG). RingCT has been supported by Monero since January 2017.

In Monero, suppose a user wants to spend m coins from his wallet, denoted as $A_s := \{(\mathrm{PK}_s^{(i)}, \mathrm{CN}_s^{(i)})\}_{i=1}^{m}$ where $\mathrm{PK}_s^{(i)}$ is the user's i-th account address and $\mathrm{CN}_s^{(i)}$ is the balance of the account. The user first chooses k output accounts $\{(\mathrm{PK}_r^{(j)}, \mathrm{CN}_r^{(j)})\}_{j=1}^{k}$ such that the sum of balances of the input accounts equals the output accounts, and sets $R := \{\mathrm{PK}_r^{(j)}\}_{j=1}^{k}$ as the output addresses. In addition, the user selects $n-1$ groups of input accounts with each containing m different accounts to anonymously spend A_s, i.e. set anonymity. Whenever receiving this transaction from the P2P blockchain network, the miners check the validity of the transaction along with its public information. The commitments are used to hide account balance. There are several special properties required for the RingCT protocol. Public keys generated by the key generation algorithm of ring signature should be homomorphic. Commitments should be homomorphic w.r.t. the same operation as public keys. Commitments to zero are well-formed public keys, each corresponding secret key of which can be derived from the randomness of commitments.

In particular, we will explore our subversion attack against the Borromean ring signature [25]. In a high-level abstraction, Borromean ring signature is a Fiat-Shamir transformation of an AND/OR composition Sigma protocol of the Schnorr's identity protocol. More specifically, let $\mathcal{P} := \{P_{i,j}\}_{i\in[0,n-1],j\in[0,m_i-1]}$ be a set of public keys, and the signer knows the secret key t_i such that $P_{i,j_i^*} = g^{t_i}$, $i \in [0, n-1]$, where j_i^* are fixed and unknown indices. For completeness, we provide the signing algorithms in Fig. 3.

2.6 Steganography

Steganography refers to techniques that allow a sender to send a message covertly over a *communication channel* so that the mere presence of the hidden message is not detectable by an adversary who monitors the channel [26, 27]. Modern steganography techniques can be applied to various media, such as im-

[1] Monero project is very active and it rapidly evolves. In fact they have two major release each year. In Oct. 2018, Monero released version 0.13.0.0 "Beryllium Bullet", which switched to Bulletproofs [22]. Since the technical specification of the latest version is not well documented yet, our work is for Monero version 0.12.0.0 and earlier versions.

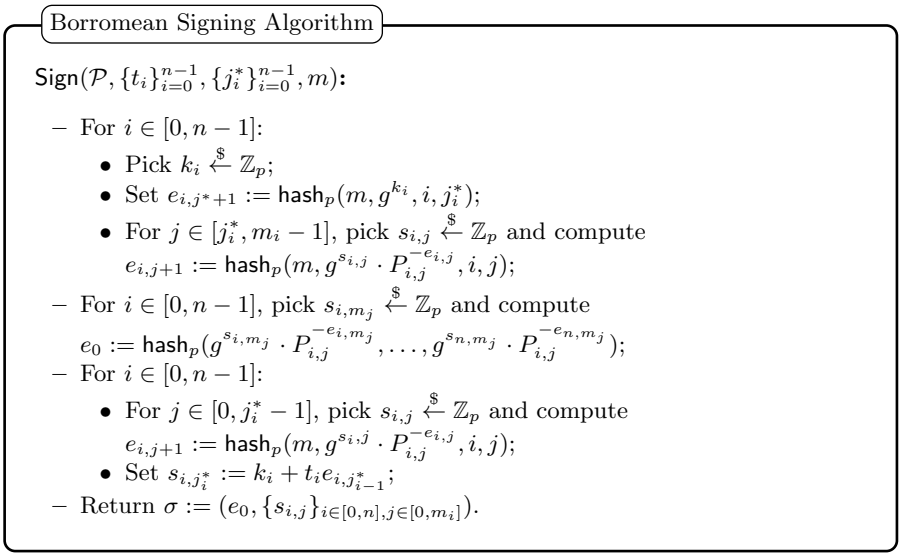

Fig. 3. Borromean Signing Algorithm.

ages, audios, HTML files, etc. A stegosystem consists of three PPT algorithms $\mathcal{ST} := (\mathsf{KeyGen}, \mathsf{Embed}, \mathsf{Extract})$ as follows:

- $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ is the key generation algorithm that takes as input the security parameter 1^λ, and it outputs an embedding key ek and an extraction key dk.
- $\mathsf{st} \leftarrow \mathsf{Embed}_{\mathcal{H},\mathsf{ek}}(m)$. Given an embedding key ek, a hidden message $m \in \{0,1\}^*$ and channel history $\mathcal{H} \in \{0,1\}^*$, Embed generates a stegotext message $\mathsf{st} \in \{0,1\}^*$ that is *indistinguishable* from the normal channel distribution $\mathcal{D}$ of innocent cover text objects ct.
- $m \leftarrow \mathsf{Extract}_{\mathsf{dk}}(\mathsf{st})$, $\mathsf{Extract}$ takes as input a extraction key dk and the stegotext $\mathsf{st} \in \{0,1\}^*$ and outputs a hidden message $m \in \{0,1\}^*$.

Definition 2 (Correctness). *We say a stegosystem* $\mathcal{ST} := (\mathsf{KeyGen}, \mathsf{Embed}, \mathsf{Extract})$ *is correct if for all* $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *we have*

$$\Pr\left[\mathsf{Extract}_{\mathsf{dk}}(\mathsf{Embed}_{\mathcal{H},\mathsf{ek}}(m))\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$

Security. The stegosystem's goal is to communicate a hidden message covertly by hiding the mere existence of the hidden communication. Therefore, a stegosystem is considered to be *secure* if an observer is not able to distinguish stegotext st from objects randomly picked from the channel distribution $\mathcal{D}$. More formally, this is defined as a *chosen hidden-text attacks* (CHA) game/experiment.

$\underline{\mathbf{Expt}^{\mathsf{CHA}}_{\mathcal{A}}(1^\lambda)}$

1. $\mathcal{A}(1^\lambda)$ outputs a message m;
2. $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$;
3. $b \leftarrow \{0, 1\}$;
4. **If** $b = 0$: $c \leftarrow \mathsf{Embed}_{\mathcal{H},\mathsf{ek}}(m)$;
   Else: $c \leftarrow \mathcal{D}$;
5. $\mathcal{A}(c)$ outputs a bit b';
6. **Return** $b \stackrel{?}{=} b'$;

We say a stegosystem $\mathcal{ST} := (\mathsf{KeyGen}, \mathsf{Embed}, \mathsf{Extract})$ is CHA-secure if

$$\mathsf{Adv}^{\mathsf{CHA}}_{\mathcal{A},\mathcal{ST}}(1^\lambda) = \left| \Pr\left[\mathbf{Expt}^{\mathsf{CHA}}_{\mathcal{A}}(1^\lambda)\right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda) \ .$$

Besides security, the following properties are also important to a stegosystem.

- *Reliability/Efficiency.* The probability that an *embedded* message is *extracted* when the stegosystem does not achieve not perfect correctness.
- *Robustness.* The inability of a challenger/warden to alter the sender's communication transcript (that contain hidden message), and possibly prevent the receiver from recovering the hidden message.

2.7 Kleptography/Algorithm-substitution attacks

Our wallet subversion attacks can be classified as kleptographic attacks [28,29] and algorithm-substitution attacks (ASA) [30,31]. The concept of subversion attacks was initially introduced by Young and Yung about two decades ago in a series of publications [28,29,32]. Recently, Bellare et al. proposed a similar type of subversion attacks called ASA attacks [28,29,32]. As a high level definition, in such attacks, the adversary maliciously tampers with the implementation of a cryptographic algorithm $\mathsf{G}_{\mathsf{IMP}}$ and changes it from its specification $\mathsf{G}_{\mathsf{SPEC}}$ algorithm, with the aim to subliminally and exclusively leak the user's secret information to the adversary while evading detection in the black-box setting. The depiction in Fig. 4 illustrates how an adversarial implementation $\mathsf{G}_{\mathsf{IMP}}$ of the algorithm $\mathsf{G}_{\mathsf{SPEC}}$ can allow the adversary, given their secret key z, to detect the subverted ciphertext c' and extract the user's secret s. Kleptographic attacks are significant due to their undetectability in the black-box setting and their severe consequences on the security of the users. See Appendix E for more details about signature subversion.

2.8 ECDSA

ECDSA is a randomized-signature scheme over the NIST elliptic curves that has been widely used in cryptocurrencies, such as Bitcoin, Ethereum, *etc.*

Elliptic Curve Over $\mathbb{F}_p$.Let $\mathsf{param} := (p, a, b, g, q, \zeta)$ be the elliptic curve parameters over $\mathbb{F}_p$, consisting of a prime p specifying the finite field $\mathbb{F}_p$, two elements $a, b \in \mathbb{F}_p$ specifying an elliptic curve $E(\mathbb{F}_p)$ defined by $E : y^2 \equiv x^3 + ax + b$

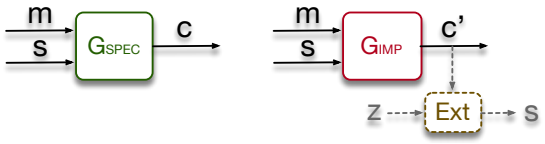

Fig. 4. Kleptography/ASA: specification $\mathsf{G}_{\mathsf{SPEC}}$ takes input as the message m and the secret s, and outputs c; whereas, the malicious implementation $\mathsf{G}_{\mathsf{IMP}}$ outputs a subverted ciphertext c' which can leak the secret s exclusively to the attacker who knows z.

ECDSA (KeyGen, Sign, Verify)

- KeyGen(param):
  - Pick random $s \leftarrow \mathbb{Z}_q$;
  - Set $S := g^s = (S_x, S_y)$;
  - Output $(\mathrm{PK} := S, \mathrm{SK} := s)$;
- Sign$(\{\mathrm{PK}\}, \mathrm{SK}, 1, m)$:
  - Pick random $r \leftarrow \mathbb{Z}_q$;
  - Set $R := g^r = (R_x, R_y)$;
  - Set $w := (\mathsf{hash}_p(m) + s \cdot R_x) \cdot r^{-1} \pmod q$;
  - Output $\sigma := (R_x, w)$;
- Verify$(\{\mathrm{PK}\}, m, \sigma)$:
  - $u_1 := \mathsf{hash}_p(m) \cdot w^{-1} \pmod q$ and $u_2 := R_x \cdot w^{-1} \pmod q$;
  - Compute $P := g^{u_1} S^{u_2} = (P_x, P_y)$;
  - Output valid if and only if $P_x = R_x \pmod q$;

Fig. 5. ECDSA Signature Scheme.

(mod p), a base point $g = (x_g, y_g)$ on $E(\mathbb{F}_p)$, a prime q which is the order of g, and an integer ζ which is the cofactor $\zeta = \#E(\mathbb{F}_p)/q$. We denote the cyclic group generated by g as $\mathbb{G}$, and it is assumed that the DDH assumption holds over $\mathbb{G}$, that is for all PPT adversary $\mathcal{A}$:

$$\mathsf{Adv}^{\mathsf{DDH}}_{\mathbb{G}}(\mathcal{A}) = \left| \Pr \begin{bmatrix} x, y \leftarrow \mathbb{Z}_q; b \leftarrow \{0,1\}; h_0 = g^{xy}; \\ h_1 \leftarrow \mathbb{G} : \mathcal{A}(g, g^x, g^y, h_b) = b \end{bmatrix} - \frac{1}{2} \right|$$

is negligible in λ.

ECDSA description. The ECDSA signature scheme is depicted in Fig. 5.

3 Generic Steganographic Attacks

Many cryptocurrencies use ring signatures to preserve users' privacy. For example, the CryptoNote framework [16], which is adopted by around 20 cryptocurrencies, uses ring signatures. As a demonstration, in this section, we describe how the *uncontrolled randomness* in CryptoNote's ring signature can be maliciously exploited. Namely, we show how the randomness within the ring signatures can

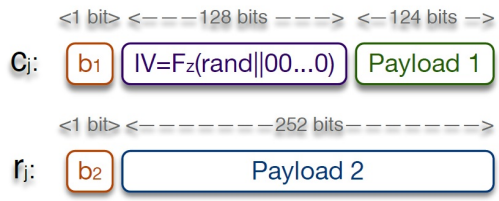

Fig. 6. Steganographic attack against CryptoNote: Format of one pair of random numbers (c_j, r_j) with 47-byte embedded stegotext.

be used to communicate covertly, store arbitrary information, and surreptitiously leak private keys. Note that the same principles are applicable to any other uncontrolled randomness in blockchain premitives.

Steganographic Attack against CryptoNote. We now describe a generic steganographic attack against all CryptoNote-based cryptocurrencies and their variants. As mentioned in Sec. 2.4, the CryptoNote protocol uses the ED25519 twisted Edwards curve, and the group order is a 253-bit prime p. The long term secret key of a user consists of two group elements $a, b \in \mathbb{Z}_p^*$, but $a := \mathsf{hash}_p(b)$ is commonly used in practical implementation. Therefore, the long term secret key of a CryptoNote account is effectively 253 bits.

As part of the one-time (linkable) ring signature, a *one-out-of-many* non-interactive zero knowledge proof is included. More specifically, for a ring of size k, the format of the ring signature is $\sigma = (I, c_1, \ldots, c_k, r_1, \ldots, r_k)$. Suppose the sender's public key is PK_i, $i \in [k]$. For all $j \in [k]$ and $j \neq i$, the components c_j and r_j are uncontrolled random group elements in $\mathbb{Z}_p$ and can be used for covert communication. (cf. Fig. 2, above) Hence, our attack is premised on steganographically embedding arbitrary information on the ring signature's random numbers (c_j, r_j).

In our attack example, $\mathsf{ek} = \mathsf{dk}$, which is a simple 128-bit random key z, is the common shared secret. The attack is explained as a three-step process carried out by two parties: a sender called Alice and a receiver called Bob.

*Step 1: embedding hidden messages (*Embed*).* As the most significant bit of a random $\mathbb{Z}_p$ element does not have uniform distribution (which is more biased to 0), to ensure (computational) indistinguishability between stegotext st and innocuous random elements $(c_j, r_j) \in \mathbb{Z}_p$, Alice embeds her secret message m in the least significant 252 bits of c_j and r_j, whereas, the most significant bits b_1 and b_2 are sampled according to the real distribution of c_j and r_j. As depicted in Fig. 6, the rest of the bits consist of a 128-bit IV, 124-bit $\mathsf{Payload}$ 1, 252-bit $\mathsf{Payload}$ 2. Let $F : \{0,1\}^{128} \times \{0,1\}^{128} \mapsto \{0,1\}^{128}$ be a block cipher that takes as input a 128-bit plaintext and a 128-bit key, and outputs a 128-bit (pseudo-random) ciphertext. Moreover, Alice uses synthetic IV to allow Bob to efficiently identify which transactions on the blockchain contain stegotext st. In particular, $\mathsf{IV} := F_z(\mathsf{rand}\|00\ldots0)$, where rand is a 64-bits random string, and $00\ldots0$ is a

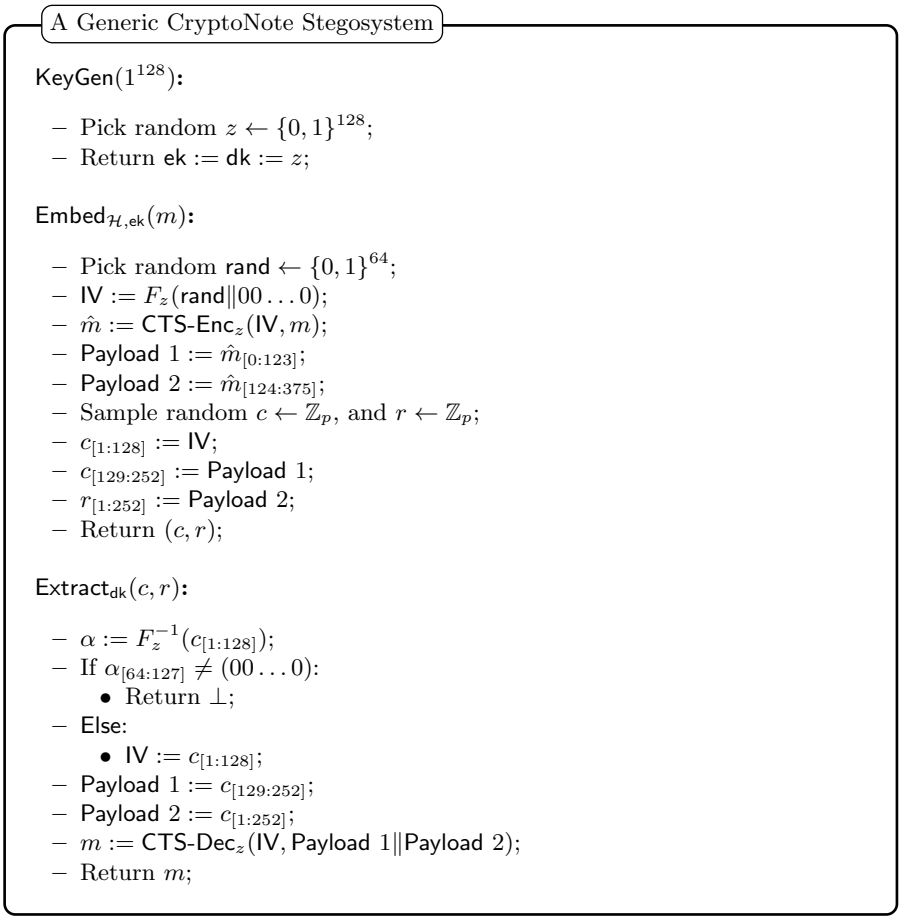

Fig. 7. Pseudo code for a generic stegosystem $\mathcal{ST}$:= (KeyGen, Embed, Extract) to covertly communicate a 376-bit message m in *one* pair of innocuous-looking (c, r), where $\lambda = 128$.

64-bit string of 0's. As a result, to check if a signature contains any st, Bob can simply try to decrypt a suspected IV, obtaining $d := F_z^{-1}(\mathsf{IV})$. If the lower half of d consists of 64 bits of 0's, then this signature contains stegotext st.

In our attack, Payload 1 and Payload 2 are jointly used to convey a 376-bit hidden message (m = Payload 1$\|$Payload 2). The payloads are encrypted via a semantically secure symmetric encryption under the secret key z and using IV. Also, to handle an arbitrary-length hidden message and ensure the resulting ciphertext has the same length as the message (besides the IV), Alice can use *Ciphertext Stealing* (CTS) as described in Appendix F.

Step 2: identifying stegotext. Unlike conventional P2P covert communication, before attempting to extract a hidden message from a transaction, Bob should first identify if the target transaction contains a stegotext st. As mentioned

before, Bob can accomplish this by parsing IV from the first two c_j's of the ring signature σ in a transaction, and checking whether the decryption of IV contains pattern 64 bits of 0's as shown in Fig. 6. Note that Embed embeds the hidden message m in one of the first two pairs of (c_j, r_j). If c_1 does not yield the IV, then Alice's secret index i must be 1, and Bob moves on to decrypt c_2 which must contain the IV, otherwise, the signature is an innocent cover text ct that does not contain st.

*Step 3: extracting hidden messages (*Extract*).* Once a steganographic ring signature is successfully identified, Bob can use the Extract algorithm to extract the hidden message. More specifically, Bob collects Payload 1 and Payload 2 as depicted in Fig. 6. Bob then uses the extraction key $\mathsf{dk} := z$ to decrypt the payload, obtaining $m := \mathsf{CTS\text{-}Dec}_z(\mathsf{IV}, \mathsf{Payload}\ 1 \| \mathsf{Payload}\ 2)$.

The pseudo code in Fig. 7 further illustrates the generic steganographic attack on CryptoNote currencies. Note that, in practice, the IV and Payload can be encrypted under two different keys derived from a single master key z. However, for notation simplicity, we use the same key here.

3.1 Security

The security of the proposed generic stegosystem against all CyptoNote-based cryptocurrencies is examined for undetectability under the chosen hidden-text attacks (CHA) game/experiment. We remark that the content-insertion techniques that use non-standard Bitcoin scripts or exchange the public key with an arbitrary string with *printable* characters, as mentioned in [1, 33], can be detected. However, our proposed steganographic attack on CryptoNote simply replaces random numbers with pseudo-random ciphers which, by definition of semantic security, are computationally indistinguishable from each other. Assuming the CTS-Enc algorithm described in Appendix F uses F as the internal PRF function, we have the following theorem.

Theorem 1. *If F is a secure pseudorandom function, the stegosystem $\mathcal{ST} :=$ (KeyGen, Embed, Extract) as shown in Fig. 7 is CHA secure.*

Proof. See Appendix A.

3.2 Robustness and Efficiency

In terms of robustness, it is easy to see that, unlike image steganography, the stegotext embedded in the signatures can never be removed while still preserving the functionality of the signatures. Therefore, there is no filter that can remove our stegotext.

Throughput. The only similar attack in literature is the proof-of-concept attack in [5] which sends a hidden message bit-by-bit through the rejection-sampling of the transaction address. Besides sending one bit of the hidden message m per transaction, their attack also sends one transaction per block. As a result, with 10 minutes to add a new block in Bitcoin, a sender needs more

than 24 hours to send a message of 20 bytes. On the other hand, our steganographic attack takes advantage of the randomness within each ring signature in CryptoNote transactions. In fact, a CryptoNote transaction contains a ring signature for each input. Therefore, if a transaction tx has y number of inputs, and n public keys in the ring of each signature, then the total number N of random numbers (c_j, r_j) in tx is $N = y * (n-1) * 2$. Whereas, the available bandwidth B in bytes is $B = 32N$. Hence, the available bandwidth in one transaction of 10 inputs and 10 public keys is more than 5KB.

Cost. Content-insertion through the use of OP_RETURN transactions and the arbitrary replacement of transaction addresses [33] render the funds unspendable. Therefore, these techniques burns funds. On the contrary, our proposed steganographic attack does not incur any additional cost, except for minimal transaction fees, as the sender can always send transactions to his own addresses. Technically, we can put arbitrarily large ring size in a transaction. In practice, we found ring size between 20 and 30 is the optimal ring size to get the transaction included quickly with minimum transaction fees. In the following section, we give some concrete costs for Bytecoin blockchain as guidelines.

4 Case studies: Bytecoin and Monero

This section contains specific implementation of the proposed attack in Sec. 3. We have implemented and evaluated the attack in two *real* cryptocurrencies; Bytecoin and Monero. Namely, we implemented the steganographic attack in the most recent release of Bytecoin (v 3.3.3) which has a market cap of around $142 millions as of the time of writing [13]. Similarly, we implemented and tested the attack in Monero which is ranked 11 among currencies and has a market cap of around $1 billion. It is important to note that as of October 2018, Monero (v 0.13.0.0) has replaced Borromean ring signatures, that is exploited by our attack, by a succinct zero-knowledge proof called Bulletproofs, which is not covered by this work. *Consequently, all of our discussion in relation with Monero is regarding previous versions of the source code mainly* (v 0.12.0.0) *and older.*

Although Monero is based on CryptoNote protocol, it uses Borromean ring signature which is different from the ring signature used in CryptoNote protocol as previously shown in Sec. 2.4. Nevertheless, our generic attack in Sec. 3 is still applicable to Monero. This emphasizes that the same attack can be extended to all public blockchain applications with randomized cryptographic primitives.

4.1 Implementation in Bytecoin

Bytecoin is an open-source cryptocurrency project [34] that implements the CryptoNote protocol described in Sec. 2.4. Accordingly, Bytecoin uses the ED25519 twisted Edwards curve and CryptoNote (linkable) ring signature to sign its transactions. As previously mentioned in Sec. 2.4, this protocol has sufficiently many uncontrolled random numbers that could be exploited to covertly communicate arbitrary information. Since Bytecoin closely follows the specifications of the

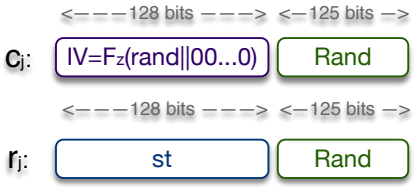

Fig. 8. Bytecoin: embedding a 16-byte *st* in one pair of (c_j, r_j) in transaction's ring signature

CryptoNote framework, it can directly be attacked using the generic steganographic attack described in Sec. 3. However, for code simplicity and clarity of demonstration, Ciphertext Stealing (CTS) is not used, and AES128 is used in the stegosystem because AES is already implemented in Bytecoin source code.

As a proof-of-concept experiment and due to ethical reasons, we only covertly transfer 16 bytes in the real-world Bytecoin without significantly abusing the blockchain system. Following the description of the generic attack in Sec. 3, we have implemented our steganographic attack on Bytecoin wallet in the following three steps.

Step 1: embedding a hidden message m and generating a signature that contains st. To embed a 16-byte hidden message m in a pair of random numbers (c_j, r_j), Alice generates a synthetic $\mathsf{IV} := \mathrm{AES}_z(\mathsf{rand} \| 00\ldots0)$ where rand is a 64-bit random string, and $00\ldots0$ is a 64-bit string of 0's. Alice then places IV as the most significant 16 bytes of c_j and sets the rest of c_j randomly. She later uses this IV along with z to generate st that is embedded in the most significant 16 bytes of r_j. Namely, $\mathsf{st} := \mathrm{AES}_z(m \oplus \mathsf{IV})$. The format of (c_j, r_j) containing st is illustrated in Fig. 8.

Furthermore, to implement this step of the attack, the Bytecoin wallet's source code is changed by mainly modifying one source file: crypto.cpp. The modified wallet simply alters the random numbers in the transaction's ring signature(s) by producing *one* pair of (c_j, r_j) as aforementioned. Note that $j \neq i$ where i is the signer's *secret* index within the ring. Particularly, the changes introduced to *crypto.cpp* affect the following two functions within the source file:

- generate_ring_signature(): This function is slightly modified to pass a counter value to the *random_scalar* function.
- random_scalar(): This function is modified by including an additional parameter in its input to specify the counter. When this counter is 0 and 1, random_scalar() generates c_j and r_j respectively which are stegotexts that hide a 16-byte message as depicted in Fig. 8.

After generating the subverted signature that contains the stegotext, the transaction is sent as per normal over the blockchain. The sender does not need to modify other parts of the wallet source code.

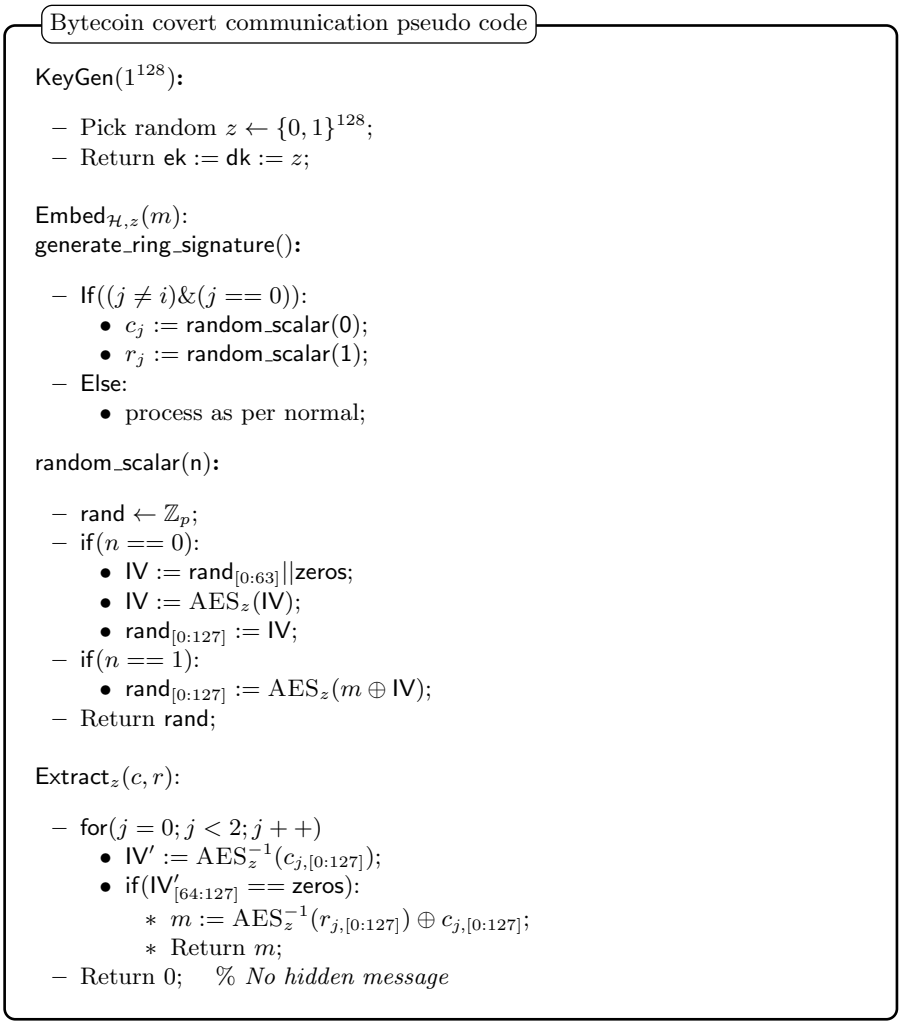

Fig. 9. Pseudo code for the implementation of covert communication in Bytecoin and similar currencies.

Step 2: identifying signature containing stegotext st. To distinguish and identify signatures containing stegotext st, Bob checks every new transaction added to the ledger. To implement this, BlockChainState.cpp is slightly modified to check each signature by decrypting each pair of (c_j, r_j) numbers. In particular, Bob uses his key z to decrypt the most significant 16 bytes of c_j to check if it contains 64 bits of zeros as in Fig. 8. If he detects such a pattern, Bob identifies the existence of a stegtext and sets IV as the most significant 16 bytes of c_j. If, however, no such pattern is detected, then the signature does not contain any hidden message.

Step 3: extracting hidden message m. After identifying a stegotext st, Bob decrypts the most signigicant 16 bytes of r_j to extract m, that is $m := \text{AES}_z^{-1}(r_{j,[0:127]}) \oplus (c_{j,[0:127]})$. This process is further clarified by the pseudo code in Fig. 9.

To further demonstrate the attack over the real Bytecoin blockchain, Appendix B provides a demo transaction included in the block at height 1671177. It contains a 16-byte hidden message "steganography".

4.2 Implementation in Monero (version 0.12.0.0)

Monero has a very complex cryptographic structure and ring signature scheme in particular. The core of Monero's wallet involves Multilayered Linkable Spontaneous Anonymous Group Signature (MLSAG) and Borromean ring signature [25]. MLSAG is similar to the 1-out-of-n ring signature that is used as part of the CryptoNote protocol; however, rather than using a ring signature on a set of n keys, MLSAG uses a ring signature on a set of n-key vectors. Using MLSAG, the signer proves to know all the private keys corresponding to one column in the public keys' matrix. Despite the massive one-time secret key, the long-term secret key is still a single group element in $\mathbb{Z}_p$.

Borromean ring signature [25], which is a generalization and based on the 1-out-of-n signature [35], is used to mask the transferred amount while enabling the receiver to know how much they have received by revealing the mask [36].

In our experiment, we chose to exploit the Borromean ring signature as it offers higher throughput. However, though with lower throughput, different primitives could also be exploited to mount steganographic attacks. Our attack on Monero is based on embedding a 32-byte hidden message m in the randomly generated $s_{i,j}$ numbers as part of the Borromean ring signature [25]. Specifically, two vectors of $s_{i,j}$ numbers are generated by the genBorromean() function: $s_{0,j}$ and $s_{1,j}$. In addition, $s_{0,j}$'s are randomly generated when the j^{th} bit commitment is 1. Two of these randomly generated $s_{0,j}$'s are used to embed m as shown in Fig. 10. In a similar manner to our attack on Bytecoin, we use AES because it is already available in the source code. More details about the implementation of the steganographic attack on Monero can be found in Appendix C.

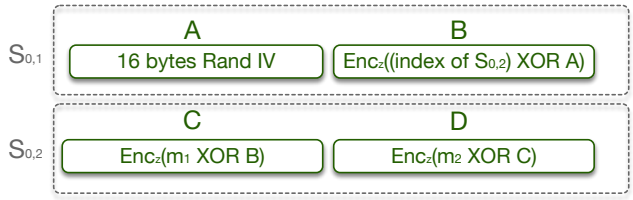

Fig. 10. Monero: embedding a 32-bybte hidden message $(m_1||m_2)$ in two random numbers $(s_{0,1}, s_{0,2})$ in the Borromean ring signature

5 Attack Scenarios: Covert Broadcast Communication and Persistent Storage

As immediate applications of our steganographic attack, adversaries can abuse the blockchain for (i) covert broadcast communication and (ii) covert persistent storage.

It is noticeable that both attack scenarios do not only facilitate malicious behaviour, but can also hinder the very future of public blockchains. In particular, if a public blockchain is known to the authorities to be abused for covert communication or storage of malicious content, then authorities in any given country may criminalize the mere participation in such blockchains. Even if participation is not criminalized, users may choose not to store the full ledger, which defeats the purpose of decentralized blockchains, and leads to a more centralized setting, where few users participate in the consensus protocol.

5.1 Attack Scenario 1: Covert Broadcast Channel

Conventional steganographic techniques typically assume that the covert communication is between two parties – a sender and a receiver. In fact, our steganographic attack can be used as a covert broadcast channel, i.e. one sender and multiple receivers. As analyzed in Sec. 3.2, to steganographically send a hidden message of 1 KB, Alice can easily craft a transaction with 4 inputs and 5 public keys. As illustrated in Fig. 11, it is easy to use our steganographic technique in conjunction with some broadcast encryption scheme, e.g. [37], to enable a practical broadcast channel. The feasibility of this attack and the high throughout illustrate the severity of this attack scenario, especially if abused by outlaws to use public blockchains as covert broadcast networks for their illicit communication.

5.2 Attack Scenario 2: Covert Data Storage and Distribution

Data storage can be viewed as a communication channel between the user and the user himself in the future. Unlike covert communication, covert persistent storage requires the uploaded content to be permanently stored and available on the blockchain. For instance, to store 1G of data, we can use transactions with a ring size of 21 with minimal transaction fees, 0.01 Bytecoin. With 2 inputs, each transaction can store about 2 kB of data, so the total cost is less than $3 given the value of Bytecoin at the time of writing. Consequently, an adversary can use Bytecoin as a *cyberlocker* and abuse the P2P network of Bytecoin as a content distribution network (CDN) forever. For example, it could be used to store pirated movies, wikileaks documents, etc.

Another special case of this scenario that shows the threat of such attack is blackmailing. An adversary, Alice, can covertly store private information about a victim, Bob. Alice may even demonstrate this to Bob by sharing the key and the extraction tool with him. Alice can then threaten Bob that she can make the information publicly available by revealing the key to everyone.

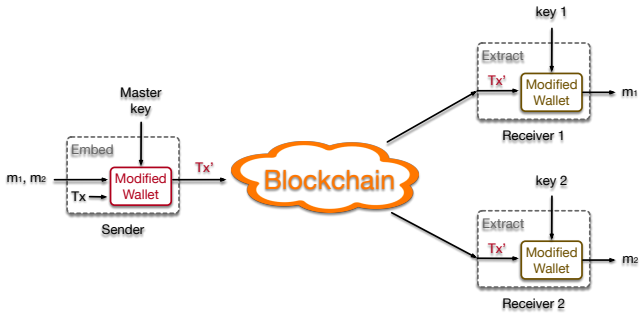

Fig. 11. Attack scenario 1: Covert broadcast communication.

6 Attack Scenario 3: Wallet Subversion

In the first two attack scenarios in Sec. 5, the sender, Alice, is complicit in the malicious attacks. This section presents another scenario where the sender is oblivious and is in fact a victim of the attack. Although this scenario may be applicable to *open-source* blockchain applications due to their complexity, it is more applicable to *close-source* and hardware-based applications, e.g. hardware wallets. The significance of this attack scenario stems from its undetectability in the black-box setting, where secrets are leaked via normal transactions posted on the blockchain, and its serious repercussions on the victim's privacy and funds.

As depicted in Fig. 12, in this scenario, Alice is an innocent user who has downloaded, or bought, a wallet that is produced by a third party Carol who has maliciously implemented the wallet. In particular, Carol used a *subversion attack* to modify a wallet and redistribute it so to leak the signer's private key, while evading detection in black-box settings. The way in which Carol modifies the wallet depends on the used cryptographic primitives and signature algorithms.

Below we present three subversion attacks that realize the scenario in Fig. 12. The first is a direct application of the generic steganographic attack described in Sec. 3 and its demo implementation in Bytecoin and Monero. We also present two more wallet subversion attacks targeting ECDSA-signature cryptocurrencies. Namely, the first attack on ECDSA-signature crypto wallets uses synthetic ephemeral key to covertly leak the entire signer's secret key over two signatures. However, it requires that the wallet is stateful in the sense that the wallet needs to store some variables from the previous signing execution. The second attack on ECDSA-signature crypto wallets is stateless and has lower throughput compared to the stateful attack. Note, in both ECDSA attacks, it is assumed that the attacker can identify the transactions generated by the victim user.

6.1 Subverting Ring-Signature Crypto Wallets

In the following we describe how the generic steganographic attack described in Sec. 3 is used by a third party, Carol, to subvert a ring-signature currency

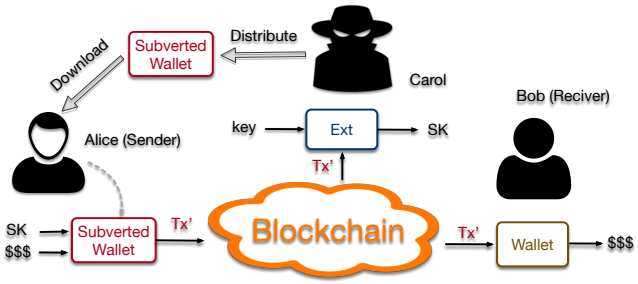

Fig. 12. Attach scenario 3: Subversion attack on crypto wallets to steal users' private keys

wallet, e.g. Bytecoin, to steal private keys. Similar attack is also applicable to Monero's Borromean signature.

Carol modifies the wallet by adding an embedding algorithm $\mathsf{Embed}_z(b)$, where b is Alice's 253-bit private key $b \in \mathbb{Z}_p$. The subverted wallet secretly executes $\mathsf{Embed}_z(b)$ to generate one pair of (c_j, r_j) as shown in Fig. 13. In this scenario, the subverted c_j contains 128-bit IV that consists of an encryption of 64 random bits rand and 64 bits of zeros, i.e. $\mathsf{IV} := F_z(\mathsf{rand} \| 00\ldots0)$. c_j also contains $\mathsf{Payload}$ 1 which is 124 bits of b. r_j contains $\mathsf{Payload}$ 2 which is 129 bits of b and $\mathsf{Payload}$ 3 which is the least significant 123 bits of Alice's public key B. The payloads are encrypted via a symmetric encryption under the same secret key z using IV.

Carol checks every added transaction for any exfilterated private keys by decrypting the first 16 bytes of c_j's from each signature, and checking if the decrypted text contains 64 bits of 0's as in Fig. 13. Note, Carol only needs to check the first two pairs of (c_j, r_j) to identify any subverted signature.

After successfully identifying a subverted signature, Carol parses and collects IV, $\mathsf{Payload}$ 1, $\mathsf{Payload}$ 2, and $\mathsf{Payload}$ 3. Carol then uses her secret key z to decrypt the payloads, obtaining $b \in \mathbb{Z}_p$ and $\mathsf{LSB}_{123}(B)$. After that she computes $a := \mathsf{hash}_p(b)$ and retrieves the corresponding public key (A, B) from the blockchain. After checking that $A = g^a$ and $B = g^b$, Carol returns the secret key $(a, b) \in (\mathbb{Z}_p)^2$. Carol can now recover all the one-time addresses and transactions and even impersonate the compromised signer, Alice, to spend her money.

6.2 Subverting ECDSA-Signature Crypto Wallets: Synthetic Randomness

Our first proposed subversion attack on ECDSA is a simplified version of the attack proposed in [38]. In this attack, we mainly aim to attack hardware wallets, and hence we assume that the users are oblivious to the adversary's secret key encapsulated in the secure hardware. The subverted algorithm is depicted in Fig. 14. Let $z, \alpha \in \mathbb{Z}_p$ be the adversary's secret keys, and set $Z := g^z$. Let $R \leftarrow \mathsf{map}(R_x)$ be a mapping function that takes as input the x-coordinate and outputs the corresponding point on the curve. The subverted wallet needs to

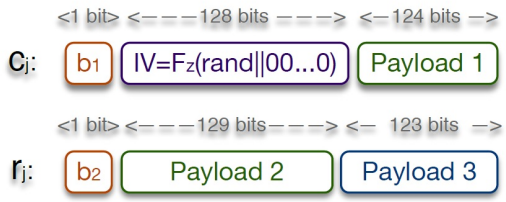

Fig. 13. Attack scenario 3: Covertly leaking the signer's private key in *one* pair of (c_j, r_j).

use algorithms $\mathsf{Sign}^{(1)}$ and $\mathsf{Sign}^{(2)}$ in turn to leak the signing key s. For the first time, $\mathsf{Sign}^{(1)}$ is identical to the original signature algorithm, however, the subtle difference is that $\mathsf{Sign}^{(1)}$ stores the ephemeral key r_1 in a long-term memory, which can be accessed during the next signature invocation. $\mathsf{Sign}^{(2)}$ is also identical to the original signature algorithm except that it deterministically generates $r_2 := \mathsf{hash}_p(g^{\alpha \cdot r_1} \cdot Z^{r_1})$, where both α and Z are hardcoded in the wallet. Once the adversary obtains two signatures σ_1, σ_2, he can use his secret keys z, α to recover the victim's signing key s. First, he parses σ_1, σ_2 as (R'_x, w_1) and (R_x, w_2). The attacker then finds the point on the curve that corresponds to R'_x, using $R' \leftarrow \mathsf{map}(R'_x)$. After that, the attacker computes $r'_2 := \mathsf{hash}_p((R')^{\alpha+z})$. Note that if r'_2 is equal to r_2 then everything is correct. Let $R := g^{r'_2} = (R_x, R_y)$. The secret key can be extracted as $s := (w_2 \cdot r'_2 - \mathsf{hash}_p(m_2)) \cdot (R_x)^{-1}$. This attack illustrates how the entire long term signing key s can be leaked exclusively to the adversary over two subverted signatures.

6.3 Subverting ECDSA-Signature Crypto Wallets: Rejection Sampling

While our first ECDSA subversion attack has a very high throughput, it has few drawbacks. First of all, it is a stateful algorithm, so it is not suitable for all scenarios, especially for software wallets. Furthermore, the first attack can only leak the signing key by the nature of its design, and not any other confidential information. Note that most cryptocurrency wallets are able to avoid the re-use of the address and signing key. As a result, the leaked signing key in our first attack, may never be used again even if the signing algorithms are executed twice with the same signing key. Nevertheless, for most wallets, there is a master key that is used to deterministically derive all the one-time signing keys.

As a result, our second subversion attack on ECDSA is stateless and is designed to leak arbitrary confidential information. As depicted in Fig. 15, the subverted signing algorithm takes as input the signing key s, the message m_i, and the secret $x \in \{0,1\}^n$ to be leaked. The signing algorithm leaks a random bit of x per signature. Let $\mathsf{PRF} : \{0,1\}^* \times \{0,1\}^\lambda \mapsto \{0,1\}^{\log n} \times \{0,1\}$ be a pseudo-random function that takes as input an arbitrary length message and the λ-bit PRF key, and it outputs a random number of $(\log n + 1)$ bits. The

Subverted ECDSA signing algorithm 1

– $\mathsf{Sign}^{(1)}(\{S\}, s, m_1)$:
  - Pick random $r_1 \leftarrow \mathbb{Z}_q$;
  - Set $R' := g^{r_1} = (R'_x, R'_y)$;
  - Set $w_1 := (\mathsf{hash}_p(m_1) + s \cdot R'_x) \cdot r_1^{-1} \pmod q$;
  - Output $\sigma_1 := (R'_x, w_1)$;

– $\mathsf{Sign}^{(2)}(\{S\}, s, m_2, r_1)$:
  - Set $r_2 := \mathsf{hash}_p(g^{\alpha \cdot r_1} \cdot Z^{r_1})$;
  - Set $R := g^{r_2} = (R_x, R_y)$;
  - Set $w_2 := (\mathsf{hash}_p(m_2) + s \cdot R_x) \cdot r_2^{-1} \pmod q$;
  - Output $\sigma_2 := (R_x, w_2)$;

– $\mathsf{Recover}(\sigma_1, \sigma_2, m_2, z, \alpha)$:
  - Set $R' \leftarrow \mathsf{map}(R'_x)$;
  - Set $r'_2 := \mathsf{hash}_p((R')^{\alpha+z})$;
  - Set $R := g^{r'_2} = (R_x, R_y)$;
  - Output $s := (w_2 \cdot r'_2 - \mathsf{hash}_p(m_2)) \cdot (R_x)^{-1}$;

Fig. 14. The subverted ECDSA signing algorithm 1.

first $\log n$ bits is interpreted as an index j, and the last 1 bit is viewed as b. The subverted signing algorithm performs a rejection-sampling to find a random $R = (R_x, R_y)$ such that $(j, b) \leftarrow \mathsf{PRF}_z(R_x)$ and $x[j] = b$. The rest signing process is identical to the original signature algorithm. Note that the rejection-sampling is efficient, and the expected repetition per signature is 1.5 times.

To recover the secret, the adversary needs to obtain a collection of the signatures generated by the subverted algorithm. We emphasize that when the secret is a master key that can be tested for its correctness, it is *not* necessary to leak the entire key in practice. Assuming the master key is 256 bits, to obtain 50% distinct key bits, the expected number of signatures is bounded by approximately 256 signatures. Asymptotically, to obtain n secret bits, we need $\theta(n \log n)$ signatures. To demonstrate this, we preformed an experiment using our rejection sampling technique to empirically test the needed number of signatures to $32, 64, 96, 128, 160, 192$ and 224 bits out of the total 256 key bits. This experiment was run 20 times to record the number of needed signatures to leak some bits of the secret key. As shown in Table 3 in Appendix D, the average number of signatures that should be intercepted by an attacker to retrieve 50% of the key, i.e. 128 bits, is about 179 signatures.

7 Countermeasures

Below, we explore the techniques that are currently found in relevant literature to mitigate the threat of arbitrary content insertion on blockchains. In addition, we propose some countermeasures to thwart the exploitation of uncontrolled randomness to launch wallet subversion attacks.

7.1 Current Content Insertion Countermeasures

In the following we summarize and discuss the current practices and techniques that can deter content insertion on blockchains:

Light Blockchains. To solve issues related with blockchain size and scalability, new blockchain designs have emerged. For example, PascalCoin [39] is a cryptocurrency that does not keep the full history of transactions but rather stores the last 100 blocks in its ledger, and actual account balances are stored in a another cryptographic structure called the SafeBox. A very similar approach is used in the mini-blockchain scheme [40] that is implemented by Cryptonite [41] which stores the actual balances in a structure called the *account tree* which is updated by the transactions in the blockchain. Because new transactions reference the *account tree* and not previous transactions in the blockchain, transactions in older blocks can be discarded. Note, older block headers are still kept in the mini-blockchain. Although these solutions are mainly proposed solve scalability issues, these new designs can deter permanent storage of malicious content.

Redactable Blockchains. Redactable blockchains have been proposed in [4] to rewrite, remove, and insert new blocks in the blockchain. In their technique, which is based on the use of Chameleon hashes [42], the redaction could be performed by a trusted central node, or a group of nodes who posses the Chameleon hash trapdoor. Similarly, $\mu chain$ [3] proposes a mutable blockchain that is based on consensus. Hence, if malicious content is identified, mutable blockchains can effectively deter the persistent storage of such content on the blockchain.

Content Filters. *Content filters* target human readable strings to detect and reject unwanted content, e.g. rejecting a transaction if its 20-Byte destination address has 18 printable characters [2].

Increasing Transaction Fees. Although increasing the transaction fees is not advisable for promoting blockchain among innocent users, and can unfairly penalize users who relay on large transactions, e.g. exchange services, minimum mandatory fees has been proposed as a countermeasure in [2] to render content insertion economically infeasible for large transactions.

Self-verifying Addresses. The goal of this technique is to deter content insertion in Bitcoin by using arbitrary addresses. The authors of [2] suggested that instead of sending an address a, c_a is sent in the transaction, where $c_a = (G^a, r, \mathsf{Sign}(G^a||r, a))$, $r = \mathsf{CRC32}(t_1||\ldots||t_i)$, and t_i is the transaction corresponding to the i^{th} input. A similar approach is to limit the address Space. For example, PascalCoin [39] has a finite address space, and accounts are limited but can be associated with any public key. Although may not be intended

Table 2. Effectiveness of Current Countermeasures Against the Three Attack Scenarios

Technique	Attack Scenarios		
	Covert Channels	Persistent Storage	Wallet Subversion
Light chains	-	✓	-
Redactable chains	-	✓	-
Content Filters	-	-	-
Increasing Transaction Fees	-	-	-
Self-verifying Addresses	-	-	-

to stop content-insertion, this practice can deter the arbitrary manipulation of transactions' addresses.

Table 2 lists all of the current countermeasures to thwart content insertion in blockchains, and compares their effectiveness against our three attack scenarios as described in Sec. 5 and Sec. 6. As seen in Table 2, light and redactable blockchains represent effective countermeasures against using blockchains to persistently store arbitrary and possibly malicious content. However, there is not currently any effective countermeasure against the use of blockchains for covert communication, nor is there any practical countermeasure against the exploitation of blockchains by Algorithm Substitution Attacks (ASA). Therefore, we urge that new blockchain designs study the use of deterministic signatures, and also the use of signature schemes that offer less redundancy which can be abused to conduct such attacks.

7.2 Wallet Subversion Countermeasures

In terms of countermeasures to wallet subversion attacks, we propose the following mitigation methods. **Containerization and Deterministic Compilation.**

Our first proposed countermeasure may be intuitive but, we believe, is also practical in certain situations. If reasonable trust can be placed on a reference executable binary, while at the same time a security-conscious user wants to compile a binary from the corresponding source code, then producing the same binary as the reference binary is a sufficient measure for insuring that the source code has not been maliciously modified. However, compiling the same source code multiple times usually results in different executable binaries due to differences in build environments. Therefore, the same build environment is needed to produce an identical binary as the reference binary.

To achieve this goal, i.e. providing an identical build environment, the developers can provide a virtual machine or a container, e.g. a *Docker* container [43], with pre-configured compiler settings and scripts to build a given source code, rendering the whole compilation process deterministic. Using this paradigm, the user can ensure the source code has not been modified by any third party after it was tested and built by the, presumably trusted, developers. This approach

Subverted ECDSA signing algorithm 2

- $\mathsf{Sign}(\{S\}, s, m_i, x)$:
  - Repeat the following process:
    - Pick random $r \leftarrow \mathbb{Z}_q$;
    - Set $R := g^r = (R_x, R_y)$;
    - Compute $(j, b) \leftarrow \mathsf{PRF}_z(R_x)$;
    - If $x[j] = b$, break the loop;
  - Set $w := (\mathsf{hash}_p(m_i) + s \cdot R_x) \cdot r^{-1} \pmod q$;
  - Output $\sigma_i := (R_x, w)$;

- $\mathsf{Recover}(\sigma_1, \ldots, \sigma_\ell, z)$:
  - Init an array $\mathcal{S} := \emptyset$;
  - For $i \in [\ell]$, do:
    - Parse σ_i as (u_i, v_i);
    - Compute $(j_i, b_i) \leftarrow \mathsf{PRF}_z(u_i)$;
    - Set $\mathcal{S}[j_i] := b_i$;
  - Output $\mathcal{S}$;

Fig. 15. The subverted ECDSA signing algorithm 2

also guarantees that there are no backdoors introduced by malicious compilers that may have not been tested by the developers. Finally, the provided container can enable multiple builders to verify the executable binary and establish trust for other users.

Signature with Synthetic Randomness. As identified by many researchers [28–31], the root of such class of subversion attacks is the *uncontrolled randomness*. One possible solution to this issue is to use deterministic signatures. In fact, all the signatures can be made deterministic using synthetic randomness. More specifically, assume a signature algorithm consumes ℓ random coins, denoted as $r_1, \ldots, r_n$. Without loss of generality, suppose the signing algorithm takes as input the signing key s and the message m. We can generate the needed random coins deterministically as $r_i := \mathsf{hash}(s, m, i)$. Based on heuristics property and onewayness of the hash function, r_i is unpredictable due to the entropy of s. On the other hand, this tweak allows offline watchdogs (verification algorithms) to compare and test an implementation with its specification.

Note that no *probabilistic polynomial time* black-box verification mechanism can ensure an implementation exactly matches its specification. This is because a malicious functionality may be triggered by a specific input, and it is impossible to verify that an implementation behaves as expected for all inputs. For instance, our attack can be modified so that the signing algorithm behaves honestly for all the inputs, except when the input message $m = m^*$ the signing algorithm switches to our attack version, where m^* is the hidden trigger that has high entropy. The elimination of such hidden triggers is discussed below.

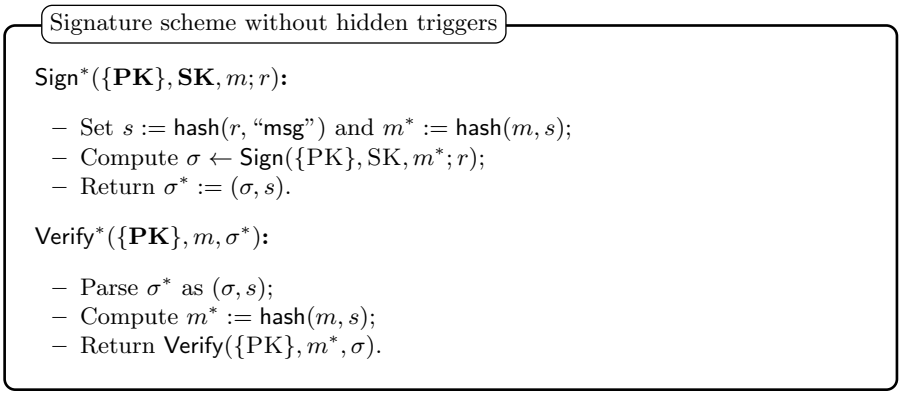

Fig. 16. Signature algorithm with hidden trigger elimination.

Instead, the offline implementation verification is only required to check polynomially many randomly sampled inputs (together with randomly sampled explicit randomness) and compare the corresponding outputs of the implementation with its specification. The most recommended approach to achieve automatic verification is to use so-called executable specifications [44]. We emphasize that the offline implementation verification algorithm must be trusted and certified. Nevertheless, it is universal and only needs simple comparison functionality, which makes it easy to ensure subversion freeness.

Hidden Trigger Elimination. As mentioned before, in practice, a subverted signature algorithm may behave maliciously when a specific input message is given. Such a specific input message has high entropy, so with negligible probability an offline watch dog can trigger and detect such a malicious behaviour. To remove hidden triggers, we need to randomize the input message. Suppose the original signature scheme consists of the following three algorithms:

– $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{KeyGen}(\mathsf{param}; r)$
– $\sigma \leftarrow \mathsf{Sign}(\{\mathrm{PK}\}, \mathrm{SK}, m \in \{0,1\}^*; r)$
– $b \leftarrow \mathsf{Verify}(\{\mathrm{PK}\}, m, \sigma)$

The proposed new signature scheme uses identical KeyGen, and the modified Sign^* and Verify^* algorithms are described in Fig. 16. We now show that the new proposed signature scheme achieves strong existential unforgeability if the original signature scheme is strong existential unforgeable under adaptive chosen-message attack.

Theorem 2. *Let* $\mathcal{S} := (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a signature scheme that achieves strong existential unforgeability under adaptive chosen-message attack. Let* $\mathsf{hash} : \{0,1\}^* \mapsto \{0,1\}^\lambda$ *be a cryptographic hash function. If* hash *securely realises a random oracle, then* $\mathcal{S}^* := (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}^*, \mathsf{Verify}^*)$ *is also strong existential unforgeable under adaptive chosen-message attack.*

Proof. See Appendix A.

External-randomness Key Generation

$\mathsf{KeyGen}^{\mathbf{D}}(\mathsf{param}; r)$:

- Set $b := \mathsf{hash}_p(r)$;
- Set $a := \mathsf{hash}_p(b)$, $A := g^a$, and $B := g^b$;
- Return PK $:= (A, B)$, TK $:= (a, B)$, and SK $:= (a, b)$.

Fig. 17. External-randomness key generation specification.

External-Randomness Architecture. Sometimes randomized signatures are desired for certain applications, the question is: *can we use randomized signatures while preventing kleptographic attacks?* Before illustrating the proposed architectural modification, let us first examine the existing running environment of a cryptocurrency wallet. It is safe to assume that the majority of users download wallets' executable binaries directly from the corresponding cryptocurrency website or a third-party software distribution platform, e.g. Apple AppStore. During the running time, depending on the functionality, the wallet may consume randomness collected by the underlying operating system (OS). For instance, Linux kernel gathers entropy from keyboard timings, mouse movements and IDE timings, and the randomness pool can be accessed via /dev/random and /dev/urandom. Although the executable binary files can be potentially subverted, unlike conventional kleptographic settings, the randomness generator is usually a part of the underlying operating system and can be trusted if we can ensure sufficient entropy. Later, we will also address untrusted OS-level randomness generator. Nevertheless, as shown in our attacks, trusted randomness source alone does not guarantee subversion-immunity, because the actual use of randomness in implementation may deviate from the corresponding software specifications. In fact, the implementation may simply bypass the specified software or hardware randomness generator. For example, in our attack, the subverted wallet uses $r' := \mathsf{Enc}_z(\mathsf{sk}; r)$ as the randomness instead of the given randomness r, where z is the adversarial key and sk is the victim's secret key to be leaked. Besides, if the randomness consumption is not restricted, the wallet can also perform rejective sampling to leak information. To control randomness usage, we propose the following modifications.

External-randomness Wallet. To ensure correct usage of randomness, we need to make a cryptocurrency wallet deterministic by externalizing the software randomness draw. At the specification level, all the algorithms of a wallet are forbidden to have any internal randomness draw component. The algorithm specifications A_{SPEC} take a λ-bit randomness as an explicit input parameter, where $\lambda \in \mathbb{N}$ is the security parameter, e.g. 256 in practice. For uniformity, all algorithms A_{SPEC} take the same amount of randomness. If more randomness is needed, they are *deterministically* derived from the input randomness r by setting $r_i := \mathsf{hash}(r, i)$, where $i \in \mathbb{N}$ is an index and hash is a cryptographically secure hash function, e.g. SHA3-256.

External-randomness Signing Algorithm

$\mathsf{Sign}(\mathsf{param}, \{P_i\}_{i\in[k]}, t_\ell, \ell, m; r)$:

- Set $I := \mathsf{hash}_g(P_\ell)$;
- Set $ctr := 0$;
- For $i \in [k]$, set $q_i := \mathsf{hash}_p(r, ctr)$ and $ctr := ctr + 1$;
- For $i \in [k], i \neq \ell$, set $w_i := \mathsf{hash}_p(r, ctr)$ and $ctr := ctr + 1$;
- For $i \in [k]$:
  - Set $L_i := g^{q_i}$ if $i = \ell$;
    Set $L_i := g^{q_i} \cdot P_i^{w_i}$ if $i \neq \ell$;
  - Set $R_i := (\mathsf{hash}_g(P_i))^{q_i}$ if $i = \ell$;
    Set $R_i := (\mathsf{hash}_g(P_i))^{q_i} \cdot I^{w_i}$ if $i \neq \ell$;
- Set $c := \mathsf{hash}_p(m, L_1, \ldots, L_k, R_1, \ldots, R_k)$;
- For $i \in [k]$:
  - Set $c_i := w_i$ if $i \neq \ell$;
    Set $c_i := c - \sum_{j=0}^{k} c_j$ if $i = \ell$;
  - Set $r_i := q_i$ if $i \neq \ell$;
    Set $r_i := q_\ell - c_\ell t_\ell$ if $i = \ell$;
- Return $\sigma := (I, c_1, \ldots, c_k, r_1, \ldots, r_k)$.

Fig. 18. External-randomness signing algorithm specification.

The main functionality of a cryptocurrency wallet is a signature scheme. Without loss of generality, a signature scheme consists of KeyGen, Sign, and Verify. (Of course, the linkable ring signature scheme used in CryptoNote also has a LNK algorithm to detect double spending, but it is deterministic.)

In the following, we modify the specification of KeyGen and Sign of CryptoNote wallet as examples. A typical key generation algorithm KeyGen takes input as the group parameter param, which defines the underlying group (or elliptic curve), e.g. Bitcoin uses SECP256K1 NIST curves and CryptoNote uses the ED25519 twisted Edwards curve. Of course, choosing a "nothing up the sleeve" group parameter is essential for subversion resilient cryptographic primitives; however, it is outside the scope of our work. We assume the commonly used blockchain group parameters are carefully examined, and are widely believed to be stego-free. The modified signature specification takes an explicit randomness $r \in \{0,1\}^\lambda$.

As shown in Fig. 17 and Fig. 18, only a few number of lines need to be modified to make a signature scheme use external randomness. The difference between the modified version and the original version is marked in red.

8 Related Work

This work is closely related to the topics of *malicious content insertion in blockchains*, *steganography*, *covert channels in blockchains*, as well as *kleptography* (a.k.a. *Algorithm-Substitution Attacks*).

Content insertion in blockchain. As far as we know, all previous research on malicious content insertion has been conducted on one application: Bit-

coin. For example, the authors of [33] provided insight regarding the various ways that could be exploited to store, possibly illegal, content onto the Bitcoin blockchain. They specifically listed four methods of embedding content in Bitcoin transactions: 1) Including up to 100 Bytes of arbitrary content in *coinbase* and OP_RETURN transactions, which offer an intended mechanism to augment transactions with arbitrary text, 2) Exchanging the public key (hash) in Pay-to-Pubkey (Pay-to-Pubkey-Hash), 3) Attaching up to 83 Bytes in *nulldata* transactions, and 4) Using non-standard scripts, e.g., by adding noneffective lines to the script. Similarly, the authors of [1] attempted to systematically analyze the non-financial content in Bitcoin's blockchain. Specifically, they surveyed the methods and services that are used to store non-financial content, and provided a general categorization of objectionable content that could be found on Bitcoin's blockchain. They found that 1.4% of all Bitcoin transactions contain non-financial data, and retrieved over 1600 files, some of which contain objectionable content, e.g. links to child pornography.

Steganography. The concept of *steganography* was introduced by Simmons' *prisoner's problem* in [45]. Simmons discussed the problem where two prisoners want to exchange secret information, an escape plan, without being detected by the prison's warden who carefully inspects the exchanged messages and will throw any suspicious communication. In this context, the problem of *steganography* is the ability of the two prisoners to communicate secret information via the warden-inspected messages without being caught.

Furthermore, Anderson et al. listed some of the limits of steganography and discussed the difficulty associated with formalizing a general proof of security for steganography [46, 47]. A number of works, e.g. [48–50], provided information-theoretic treatment of steganography security and robustness. The authors of [51] used information theory to model the security of stegosystems by the relative entropy (uncertainty) of the hidden message m, the entropy of the stegotext st and the entropy of normal cover text. In a similar manner, [50] used information theory to quantify the security of a stegosystem as the discrimination between the distribution of innocent cover text and that of stegotext. More recently, Hopper et al. provided a cryptographic formalization of steganographic security and robustness [26]. In addition, they presented a definition for the security of a steganographic system in terms of the *computational* indistinguishability of stegotext from cover text.

Covert channels in blockchains. While there is a relatively significant body of research on *content insertion* in Bitcoin's blockchain [1, 2, 33, 52], the authors of [5] were the first to discuss the use of *steganography* to *covertly* communicate in Bitcoin's blockchain. However, due to its limitation, the authors of [5] consider their attack to be a proof of concept rather than a practical attack. In short, to covertly send a hidden message m using the attack from [5], the sender sends $\lambda = \delta||m'$, where m' is the symmetric cipher encrypted under the key k, i.e. $m' = \mathrm{Enc}_k(m)$, and δ is a starting pattern known to both the sender and the receiver(s). In addition, the string λ is sent bit by bit through the rejection-sampling of the transaction address a, so that the least significant bit of the

address a is the same as the i^{th} bit of λ, i.e. $a[\mathsf{LSB}] = \lambda[i]$. Only one bit is sent in every transaction, and, to maintain order, only one transaction is sent in each block. Therefore, with average time of 10 minutes to validate a block, the sender needs more than 24 hours to send a message of 20 bytes. Moreover, the receiver continuously checks the payments generated by the sender and if he ever detects δ, he retrieves m' bit by bit and decrypts it to reveal m, i.e. $m = \mathrm{Dec}_k(m')$. Note that because the receiver needs to know the sender's identity, this method is not usable in blockchains where the sender's identity is anonymous, e.g. Zcash.

Kleptography and ASA. Our wallet subversion attack falls within the realm the topic of *Algorithm-Substitution Attacks (ASA)* [30,31], also called *Kleptography* [28,29] and *Subversion Attacks* (SA) [53]. The notion of Kleptography was introduced by Young and Yung in 1996 [28,29]. Subsequent work demonstrated the possible use of ASA in mass surveillance, and the susceptibility of all randomized symmetric encryption schemes to such attacks [31, 54]. Another demonstration of ASA attacks is found in the work of Goh et al. [55] who presented practical hidden key-recovery attacks against the SSL/TLS and SSH2 protocols by modifying the implementation of the OpenSSL library. In the context of signature schemes, Young and Yung [32] showed that DSA signature schemes can be subverted to leak secret information. Also, Ateniese et al. [53] described stateful subversion attacks against randomized coin-injective and coin-extractable signature schemes, and explained how their attacks can subvert a randomized signature even if it has only one bit of randomness. They also proposed subversion-resilient signature schemes using so-called cryptographic reverse firewalls to sanitize the signatures generated by untrusted algorithm implementations. However, their solution cannot be generalized to mitigate the risk of subversion for the vast emerging blockchain technology protocols. In addition, Russell et al. [56] modeled and proved a full domain hash-based signature scheme achieves subversion resilience. Recently, Russell et al. [57] proposed the use of a splitting-randomness technique to secure a randomizable IND-CPA secure public key encryption. But it is unknown how to apply their technique in the blockchain context with reasonable assumptions.

9 Conclusion and Future Work

The main aim of this work is to highlight the potential threat of maliciously abusing uncontrolled randomness in randomized cryptographic primitives in blockchain applications. To illustrate the idea, we designed, implemented, and evaluated our attacks against the widely-used ECDSA signature scheme, the ring signature used in the CryptoNote framework, and the Ring Confidential Transaction used in Monero (up to version 0.12.0.0). We emphasize that this line of research is far from being completed, and our technique can also be used on any other randomized cryptographic primitives, such as SNARK and Bulletproofs. We plan to extend our technique to non-interactive zero-knowledge proofs/arguments. Finally, with regard to wallet subversion attacks, we hope that our work motivates the research of subversion-resistant randomized crypto-

graphic primitives and encourages novel software design principles that thwart unintentional and intentional implementation of vulnerabilities and backdoors.

References

1. R. Matzutt, J. Hiller, M. Henze, J. H. Ziegeldor, D. Mullman, O. Hohlfeld, and K. Wehrle, "A quantitative analysis of the impact of arbitrary blockchain content on bitcoin," in *FC 2018*, 2018.
2. R. Matzutt, M. Henze, J. H. Ziegeldorf, J. Hiller, and K. Wehrle, "Thwarting unwanted blockchain content insertion," in *IC2E 2018*, pp. 364–370, April 2018.
3. I. Puddu and A. Dmitrienko, "μchain: How to forget without hard forks," *IACR Cryptology ePrint Archive 2017/106*, 2017. https://eprint.iacr.org/2017/106.
4. G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable blockchain or rewriting history in bitcoin and friends," in *Euro S&P 2017*, pp. 111–126, April 2017.
5. J. Partala, "Provably secure covert communication on blockchain," *Cryptography*, vol. 2, no. 3, 2018.
6. G. Fuchsbauer, "Subversion-zero-knowledge snarks," in *PKC 2018*, pp. 315–347, 2018.
7. B. Abdolmaleki, K. Baghery, H. Lipmaa, and M. Zajac, "A subversion-resistant snark," in *ASIACRYPT 2017*, pp. 3–33, 2017.
8. J. Knockel, T. Ristenpart, and J. R. Crandall, "When textbook RSA is used to protect the privacy of hundreds of millions of users," *CoRR*, vol. abs/1802.03367, 2018.
9. L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *CCS 2016*, pp. 254–269, 2016.
10. D. Siegel, "Understanding the dao attack," 2016. Available Online: https://www.coindesk.com/understanding-dao-hack-journalists (Last accessed 7-Feb-2018).
11. S. Azouvi, M. Maller, and S. Meiklejohn, "Egalitarian society or benevolent dictatorship: The state of cryptocurrency governance," in *5th Workshop on Bitcoin and Blockchain Research*, 2018.
12. Giza Device Ltd, "Giza wallet," 2017. Avialable Online: https://www.gizadevice.com/ (Last accessed 7-Feb-2018).
13. CoinMarketCap, "Cryptocurrency market capitalizations," 2018. Available Online: https://coinmarketcap.com/ (Last accessed 26-Nov-2018).
14. S. Nakamoto, "A Peer-to-Peer Electronic Cash System," 2008. Available Online: https://bitcoin.org/bitcoin.pdf (Last accessed 05-Nov-2018).
15. A. Bender, J. Katz, and R. Morselli, "Ring signatures: Stronger definitions, and constructions without random oracles," *J. Cryptol.*, vol. 22, pp. 114–138, Dec. 2008.
16. N. V. Saberhagen, "Cryptonote v 2.0," 2013. whitepaper, Available online: https://cryptonote.org/whitepaper.pdf, (Last accessed 23-Nov-2018).
17. Bytecoin Org., "Bytecoin (bcn)," 2018. Available Online: https://bytecoin.org/ (Last accessed 23-Nov-2018).
18. CryptoNote Org., "Cryptonotecoin," 2018. Available Online: http://cryptonote-coin.org/ (Last accessed 23-Nov-2018).
19. Fantomcoin, "Fantomcoin," 2014. Available Online: http://fantomcoin.org/ (Last accessed 23-Nov-2018).

20. Monero, "Monero," 2018. Available Online: https://getmonero.org/ (Last accessed 07-Feb-2018).
21. R. Spagni, "Monero project github repository," 2018. Available Online: https://github.com/monero-project/monero (Last accessed 07-Feb-2017).
22. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *S&P 2018*, vol. 00, pp. 319–338, 2018.
23. S. Noether, "Ring signature confidential transactions for monero." Cryptology ePrint Archive, Report 2015/1098, 2015.
24. T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO '91*, pp. 129–140, 1992.
25. G. Maxwell and A. Poelstra, "Borromean Ring Signatures," 2015. Available Online: http://diyhpl.us/~bryan/papers2/bitcoin/Borromean%20ring%20signatures.pdf (Last accessed 07-Feb-2018).
26. N. J. Hopper, J. Langford, and L. von Ahn, "Provably secure steganography," in *CRYPTO 2002*, 2002.
27. N. Dedić, G. Itkis, L. Reyzin, and S. Russell, "Upper and lower bounds on black-box steganography," *Journal of Cryptology*, vol. 22, pp. 365–394, Jul 2009.
28. A. Young and M. Yung, "The dark side of "black-box" cryptography or: Should we trust capstone?," in *CRYPTO '96*, 1996.
29. A. Young and M. Yung, "Kleptography: Using cryptography against cryptography," in *EUROCRYPT '97*, 1997.
30. M. Bellare, K. G. Paterson, and P. Rogaway, "Security of symmetric encryption against mass surveillance," in *CRYPTO 2014*, (Berlin, Heidelberg), pp. 1–19, Springer Berlin Heidelberg, 2014.
31. M. Bellare and J. Jaeger, "Mass-surveillance without the State : Strongly Undetectable Algorithm-Substitution Attacks," in *CCS*, 2015.
32. A. Young and M. Yung, "The prevalence of kleptographic attacks on discrete-log based cryptosystems," in *CRYPTO '97*, 1997.
33. R. Matzutt, O. Hohlfeld, M. Henze, R. Rawiel, J. H. Ziegeldorf, and K. Wehrle, "Poster: I don't want that content! on the risks of exploiting bitcoin's blockchain as a content store," in *CCS '16*, 2016.
34. B. D. Team, "Bytecoin project github repository," 2018. Available Online: https://github.com/bcndev (Last accessed 26-Nov-2018).
35. M. Abe, M. Ohkubo, and K. Suzuki, "1-out-of-n signatures from a variety of keys," in *ASIACRYPT 2002*, 2002.
36. G. Maxwell, "Confidential Transactions," 2018. Available Online: https://people.xiph.org/~greg/confidential_values.txt (Last accessed 07-Feb-2018).
37. A. Fiat and M. Naor, "Broadcast encryption," in *CRYPTO' 93* (D. R. Stinson, ed.), (Berlin, Heidelberg), pp. 480–491, Springer Berlin Heidelberg, 1994.
38. E. Mohamed and H. Elkamchouchi, "Kleptographic Attacks on Elliptic Curve Cryptosystems," *Journal of Computer Science*, vol. 10, no. 6, pp. 213–215, 2010.
39. A. Molina and H. Schoenfeld, "PascalCoin Whitepaper v2," 2017. Available Online: https://www.pascalcoin.org/PascalCoinWhitePaperV2.pdf (Last accessed 08-November-2018).
40. J.D. Bruce, "The Mini-Blockchain Scheme - Rev. 3," 2017. Available Online: http://cryptonite.info/files/mbc-scheme-rev3.pdf (Last accessed 08-November-2018).
41. Mini-blockchain Project, "Cryptonite Cryptocurrency," 2018. Available Online: http://cryptonite.info/ (Last accessed 01-November-2018).

42. J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig, "Chameleon-hashes with ephemeral trapdoors," in *PKC 2017*, 2017.
43. Docker-Inc., "Docker," 2018. Available Online: https://www.docker.com (Last accessed 7-Feb-2018).
44. M. Cardinal, *Executable Specifications with Scrum*. Upper Saddle River, NJ: Addison-Wesley, 2014.
45. G. J. Simmons, *The Prisoners' Problem and the Subliminal Channel*, pp. 51–67. Boston, MA: Springer US, 1984.
46. R. Anderson, "Stretching the limits of steganography," in *Information Hiding*, pp. 39–48, Springer Berlin Heidelberg, 1996.
47. R. J. Anderson and F. A. P. Petitcolas, "On the limits of steganography," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 474–481, May 1998.
48. J. A. O'Sullivan, P. Moulin, and J. M. Ettinger, "Information theoretic analysis of steganography," in *Proceedings. 1998 IEEE International Symposium on Information Theory*, 1998.
49. T. Mittelholzer, "An information-theoretic approach to steganography and watermarking," in *Information Hiding*, 2000.
50. C. Cachin, "An information-theoretic model for steganography," in *Information Hiding*, 1998.
51. J. Zöllner, H. Federrath, H. Klimant, A. Pfitzmann, R. Piotraschke, A. Westfeld, G. Wicke, and G. Wolf, "Modeling the security of steganographic systems," in *Information Hiding*, 1998.
52. K. Shirriff, "Hidden surprises in the bitcoin blockchain and how they are stored: Nelson mandela, wikileaks, photos, and python software," 2014. Available Online: http://www.righto.com/2014/02/ascii-bernanke-wikileaks-photographs.html (Last accessed 01-November-2018).
53. G. Ateniese, B. Magri, and D. Venturi, "Subversion-resilient signature schemes," in *CCS '15*, pp. 364–375, 2015.
54. M. Bellare and V. T. Hoang, "Resisting randomness subversion: Fast deterministic and hedged public-key encryption in the standard model," in *EUROCRYPT 2015*, 2015.
55. E.-J. Goh, D. Boneh, B. Pinkas, and P. Golle, "The design and implementation of protocol-based hidden key recovery," in *Information Security*, 2003.
56. A. Russell, Q. Tang, M. Yung, and H.-S. Zhou, "Cliptography: Clipping the power of kleptographic attacks," in *ASIACRYPT*, 2016.
57. A. Russell, Q. Tang, M. Yung, and H.-S. Zhou, "Destroying steganography via amalgamation: Kleptographically cpa secure public key encryption." Cryptology ePrint Archive, Report 2016/530, 2016.
58. J. Katz and V. Vaikuntanathan, "Signature schemes with bounded leakage resilience," in *ASIACRYPT 2009*, 2009.
59. E. Boyle, G. Segev, and D. Wichs, "Fully leakage-resilient signatures," in *EUROCRYPT 2011*, 2011.
60. M. J. Dworkin, "Recommendation for block cipher modes of operation: Three variants of ciphertext stealing for cbc mode." NIST Pubs, Report Number 800-38A Addendum, 2010. Available Online: https://www.gpo.gov/fdsys/pkg/GOVPUB-C13-c0b0bae5f66880bf051f6d4ac2d8f07d/pdf/GOVPUB-C13-c0b0bae5f66880bf051f6d4ac2d8f07d.pdf (Last accessed 08-Feb-2018).

Appendices

A Security Proofs

Proof of Theorem 1

In this section, we provide a full proof of Theorem 1.

Proof. We prove this theorem by reduction. Assume there exists a PPT adversary $\mathcal{A}$ who can break $\mathcal{ST}$ with an non-negligible $\mathsf{Adv}^{\mathsf{CHA}}_{\mathcal{A},\mathcal{ST}}(1^\lambda)$ advantage w.r.t. the CHA game. We need to construct a PPT adversary $\mathcal{B}$ who can break the PRF game for F. During the reduction game, $\mathcal{B}$ plays as a challenger for $\mathcal{A}$ in the CHA game. Upon receiving m from $\mathcal{A}$, $\mathcal{B}$ picks random $\mathsf{rand} \leftarrow \{0,1\}^{64}$ and sets $x := \mathsf{rand} \| 00 \ldots 0$. $\mathcal{B}$ then queries x to the PRF game challenger and obtains IV. Subsequently, $\mathcal{B}$ queries $\mathsf{IV}, \mathsf{IV}+1, \mathsf{IV}+2$ to the PRF game challenger, and obtains k_1, k_2, k_3. $\mathcal{B}$ then compute c_1, c_2, c_3 according to the description shown in Fig. 21. It then computes (c, r) as described in Fig. 7. $\mathcal{B}$ flips a coin $b \leftarrow \{0,1\}$. If $b = 0$, $\mathcal{B}$ computes a ring signature using (c, r); otherwise, $\mathcal{B}$ computes a ring signature normally. $\mathcal{B}$ then sends the resulting signature to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a guess b'. Assume the challenge bit in the PRF game is β, i.e. $\beta = 0$ is in the PRF mode; $\beta = 1$ is in the random function mode. If $b = b'$, $\mathcal{B}$ outputs $\beta^* = 0$; otherwise, $\mathcal{B}$ outputs $\beta^* = 1$.

$$
\begin{aligned}
\Pr[\mathcal{B} \text{ win}] &= \Pr[\beta^* = 0 | \beta = 0] \cdot \Pr[\beta = 0] + \\
&\quad + \Pr[\beta^* = 1 | \beta = 1] \cdot \Pr[\beta = 1] \\
&= \Pr\left[\mathbf{Expt}^{\mathsf{CHA}}_{\mathcal{A}}(1^\lambda)\right] \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} \\
&= (\mathsf{Adv}^{\mathsf{CHA}}_{\mathcal{A},\mathcal{ST}}(1^\lambda) + \frac{1}{2}) \cdot \frac{1}{2} + \frac{1}{4} \\
&= \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{CHA}}_{\mathcal{A},\mathcal{ST}}(1^\lambda) + \frac{1}{2}
\end{aligned}
$$

Hence, the advtantage of $\mathcal{B}$ w.r.t to the PRF game is

$$
\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B},F} = \left| \Pr[\mathcal{B}] \text{ win} - \frac{1}{2} \right| = \frac{1}{2} \cdot \mathsf{Adv}^{\mathsf{CHA}}_{\mathcal{A},\mathcal{ST}}(1^\lambda) \ .
$$

Since $\mathsf{Adv}^{\mathsf{CHA}}_{\mathcal{A},\mathcal{ST}}(1^\lambda)$ is non-negligible, we have $\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B},F}$ is also non-negligible, which concludes the proof.

Proof of Theorem 2

In this section, we provide a full proof of Theorem 2.

Proof. We proof this theorem by reduction. Assume there exists a PPT adversary $\mathcal{A}$ who can break $\mathcal{S}^* := (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}^*, \mathsf{Verify}^*)$ with at most q signature queries and at least ε advantage. We need to construct a PPT adversary

Demo Steganographic Bytecoin Transaction

- **Block height:** 1671177
- **Transaction id:** 52caba6ef4e4716ac8a25681eb3f380d3d1fee057ada7eb62d687af36f1a44ff
- **Sender's address:** 26c6YmZmLJZYxnVAt56kRraBhxiEUt8yoJR3VV4UV5VcRM9Pzs5qV7KStQHaa7xkAHej3WTTxtAc1KHbCSPoZ2ms3bdUsY6.
- **Receiver's address:** 26c6YmZmLJZYxnVAt56kRraBhxiEUt8yoJR3VV4UV5VcRM9Pzs5qV7KStQHaa7xkAHej3WTTxtAc1KHbCSPoZ2ms3bdUsY6.
- **Mixin count (ring size):** 6

Fig. 19. An example of the stegangraphically-generated bytecoin transaction

$\mathcal{B}$ who can break the $\mathcal{S} := (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. During the reduction game, upon receiving query m_i from $\mathcal{A}$, $\mathcal{B}$ computes $m_i^* := \mathsf{hash}(m_i, s_i)$ with a randomly sampled $s_i \in \{0,1\}^\lambda$, and then query the signing oracle with m_i^*. After getting σ_i back from the signing oracle, $\mathcal{B}$ return $\sigma_i^* := (\sigma_i, s_i)$ to $\mathcal{A}$. Whenever $\mathcal{A}$ can provide a valid pair $(\hat{m}, \hat{\sigma})$ as forgery $(\hat{\sigma} = (\sigma', s'))$, since the probability that $\mathcal{A}$ can find a collision on hash is negligible, $\mathcal{B}$ can compute $m' := \mathsf{hash}(\hat{m}, s')$. By definition, $(\hat{\sigma}, m')$ is a forgery against $\mathcal{S}$.

B Demo Bytecoin Steganographically Created Transaction

The attack described in Sec. 4 has been executed, and the transaction with the attributes shown in Fig. 19 has been generated. To demonstrate how the hidden message is embedded and extracted we provide an extraction tool that can be downloaded and tested from: https://github.com[2]. The repository also contains the actual transaction binary in tx.txt and includes a pair (c, r) of random numbers containing a 16-byte hidden message, where (c, r) is found in cr.txt. The transaction hash in Fig. 19 can also be seen in any Bytecoin explorer, like https://minergate.com/blockchain/bcn/blocks, and the provided transaction binary should hash to the same hash value.

C Detailed Implementation of Steganographic Attack in Monero

The following is the details of our implementation of the generic steganographic attack in Monero (v0.12.0.0) and older).

[2] Actual GitHub repository is omitted for blind review

Steganographic Monero Transaction

- **Block height:** 1502164
- **Transaction id:** $e4b7982b081a17892525f1b1d3011ec06a0820cbf451d3a64f8ea998104a753c$
- **Sender's address:** $455Bu1zXzgXEeXxrjzRSsEifP8WgtLTKYLreQ7RrA1fcFi2UKjgtc2UBapB9AcDaitdY7SdWGFsEZRELL8A1nMnEFRVZg47$.
- **Receiver's address:** $42F5itWciYAg5QJxZEqWz5hrQNFaySUbxfxsjcdp8FnrRM68c8Nzujm3UqfscVC6r2c2GwuiP4sRsQv3ZZUc1spjUHuDHsx$.
- **Mixin count (ring size):** 5

Fig. 20. An example of the steganographically-generated Monero transaction

Step 1: embedding a hidden message m and generating a signature that contains st. The sender's wallet is modified to surreptitiously embed a 32-byte message m in the randomly generated $s_{i,j}$ numbers as part of the Borromean ring signature. Specifically, two vectors of $s_{i,j}$ numbers are generated by the genBorromean() function: $s_{0,j}$ and $s_{1,j}$. In addition, $s_{0,j}$'s are randomly generated when the j^{th} bit commitment is 1, and two of these randomly generated $s_{0,j}$'s are used in our attack to embed the stegotext st. For simplicity, we use $s_{0,1}$ and $s_{0,2}$ to denote the first two randomly generated numbers in $s_{0,j}$ vector, although they might not necessarily correspond to $j = 1$ and $j = 2$ respectively.

Fig. 10 shows the two subverted numbers, in which $s_{0,1}$ includes 16 bytes of random IV concatenated with 1 byte representing the index of $s_{0,2}$ and 15 bytes of zeroes, where the last 16 bytes are sent encrypted using AES-CBC. The second subverted random number, $s_{0,2}$, contains hidden message m encrypted using AES-CBC under the key z.

This step of the attack is achieved by slightly modifying two functions: genBorromean() and skGen() in two files: rctSig.cpp and rctOps.cpp. genBorromean() is modified to pass two extra parameters to skGen(). The first parameter is the counter that indicates which of the two random numbers is to be generated, while the second parameter represents the index of j^{th} bit that corresponds to the second number $s_{0,2}$ within the $s_{0,j}$ vector. When the value of the counter is 0 or 1, skGen() generates random numbers according to Fig. 10, otherwise executes as normal.

Step 2: identifying signature containing stegotext st, and extracting hidden message m. To identify transactions containing st, the source blockchain.cpp file is modified to check the randomness within each new transaction and identify signatures containing stegotext. The receiver tests each number in the $s_{0,j}$ vector by looking for a random IV that decrypts the second half of the tested number to a similar pattern as $s_{0,1}$ in Fig. 10. Once this pattern is detected, the receiver concludes that this signature contains st and retrieves the index of $s_{0,2}$ from the 16^{th} byte of $s_{0,1}$.

Table 3. Number of signatures needed to leak bits of the long-term private key in our rejection-sampling ECDSA subversion attack

	Number of Signatures to Leak Key Bits						
Exp. #	**32**	**64**	**96**	**128**	**160**	**192**	**224**
1	35	71	117	173	227	314	520
2	33	73	119	189	253	344	485
3	32	74	120	170	230	339	471
4	34	70	114	179	238	335	491
5	32	69	119	189	280	385	552
6	32	76	122	170	233	333	526
7	36	76	120	180	262	400	576
8	32	71	127	197	259	368	566
9	33	72	115	162	242	348	528
10	32	71	121	177	243	363	498
11	31	70	121	180	246	345	524
12	35	76	120	181	260	386	563
13	33	69	124	190	251	352	506
14	32	72	121	180	255	353	518
15	33	78	124	178	246	355	539
16	34	72	113	168	232	331	522
17	35	72	111	162	228	340	512
18	31	79	125	201	281	378	544
19	37	76	122	174	243	329	475
20	34	75	121	175	260	369	570
Average	33.3	73.1	119.8	178.8	248.4	353.4	524.3
Std. dev.	1.6	2.9	3.9	10.2	15.2	21.6	30.5

When a malicious signature is detected, the receiver retrieves the index of $s_{0,2}$ as above. The receiver then extracts the hidden message by decrypting $s_{0,2}$ using his key z with AES-CBC. Fig. 20 shows a Monero transaction that has been steganographically subverted by our attack, and has been successfully posted to the Monero blockchain

D ECDSA-Signature Rejection-Sampling Experiment

Table 3 illustrates the experimental results on how many signatures needed to obtain $32, 64, 96, 128, 160, 192$ and 224 bits out of the total 256 key bits. This experiment was run 20 times to record the number of needed signatures to leak some bits of the secret key. As seen in Table 3, the average number of signatures that should be intercepted by the attacker to retrieve 50% of the key, i.e. 128 bits, is about 179 signatures.

E Signature subversion

In theory, the $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}$ algorithms of a signature scheme can be subverted to leak secret information. However, in practice most blockchain platforms do not generate the setup parameters themselves; instead, widely trusted setup parameters, such as in ED25519, are adopted. Therefore, we don't consider Setup algorithm in this work. In terms of KeyGen algorithms, they are usually based on some one-way function, and it is possible to leak $O(\log \lambda)$ bits through rejective sampling. Nevertheless, for most signature schemes, this would not be sufficient to allow the adversary to forge a signature. See *leakage resilient signatures* in [58, 59] for more discussion. Therefore, this work focuses on the subversion of the Sign algorithm. As a result, we adopt the following modified definition of undetectability from [53].

Public/Secret Undetectability. The undetectability is used to model the fact that normal users cannot distinguish if a signature is produced by a subverted signing algorithm or the genuine one.

Definition 3. *Let* $\mathcal{S} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be a signature scheme. Let* $\mathcal{M}$ *be the message space. We say a subverted* Sign^* *algorithm is secretly undetectable w.r.t.* $\mathcal{S}$ *if for all PPT adversary* $\mathcal{A}$ *we have any* $\{(\mathrm{PK}_i, \mathrm{SK}_i)\}_{i=1}^n$ *output by* $\mathsf{KeyGen}(\mathsf{param})$ *for any integer* $\lambda \in \mathbb{N}$*, any* $n = \mathsf{poly}(\lambda)$*, any* $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$*, any* $\{(\mathrm{PK}_i, \mathrm{SK}_i)\}_{i=1}^n$ *output by* $\mathsf{KeyGen}(\mathsf{param})$*, any* $\ell \in [n]$*, we have:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SU}}(1^\lambda) = \left| \Pr\left[\mathbf{Expt}_{\mathcal{A}}^{\mathsf{SU}}(1^\lambda)\right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda)$$

w.r.t. the following game/experiment:

$\mathbf{Expt}_{\mathcal{A}}^{\mathsf{SU}}(1^\lambda)$

1. *Pick* $b \leftarrow \{0, 1\}$*;*
2. *Send* $(\{\mathrm{PK}_i\}_{i=1}^n, \mathrm{SK}_\ell)$ *to* $\mathcal{A}$*;*
3. ***For*** $j \in \{1, \ldots, k\}$*,* $\mathcal{A}$ *queries* $m_j \in \mathcal{M}$ *and obtains*
   - $\sigma_j \leftarrow \mathsf{Sign}(\{\mathrm{PK}_i\}_{i=1}^n, \mathrm{SK}_\ell, \ell, m_j)$ *if* $b = 0$*;*
   - $\sigma_j^* \leftarrow \mathsf{Sign}^*(\{\mathrm{PK}_i\}_{i=1}^n, \mathrm{SK}_\ell, \ell, m_j)$ *if* $b = 1$*;*
4. $\mathcal{A}$ *outputs a bit* b'*;*
6. ***Return*** $b \stackrel{?}{=} b'$*;*

We say a subverted Sign^* *algorithm is publicly undetectable w.r.t.* $\mathcal{S}$ *if in step 1 of the above game* $\mathcal{A}$ *only receives* $\{\mathrm{PK}_i\}_{i=1}^n$*.*

F Ciphertext Stealing Technique

In our attack, the leaked information is encrypted by a semantically secure symmetric-key encryption scheme. To minimize the number of lines of code to be changed, we need to adopt a readily implemented encryption algorithm. In

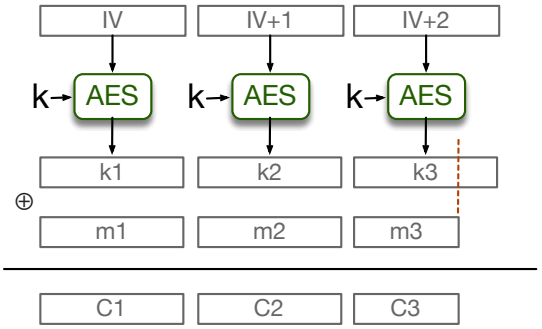

Fig. 21. Ciphertext Stealing (CTR mode)

our experiment, both Bytecoin and Monero wallets already have AES-128 algorithm, which; therefore, can be used as a building block of the semantically secure encryption. However, the message length is usually not a perfect multiple of 128 bits. To maximize the subversion channel capacity, one option is to adopt the concept of Ciphertext Stealing (CTS) [60]. For any given message length, with ciphertext stealing techniques, the ciphertext length is exactly the same as the message length (besides the IV). Our generic attack, in Sec. 3, uses CTR-mode based ciphertext stealing technique, where the last encryption block is truncated to fit the message length. The encryption algorithm $\mathsf{CTS\text{-}Enc}_k(\mathsf{IV}, m)$ is depicted in Fig. 21. We refer interested readers to [60] for more operation modes with CTS, such as CBC.