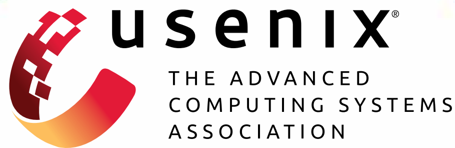

Engineering a backdoored bitcoin wallet

Adam Scott and Sean Andersen, *Block, Inc.*

https://www.usenix.org/conference/woot24/presentation/scott

This paper is included in the Proceedings of the
18th USENIX WOOT Conference on Offensive Technologies.

August 12–13, 2024 • Philadelphia, PA, USA

ISBN 978-1-939133-43-4

Open access to the
Proceedings of the 18th USENIX WOOT
Conference on Offensive Technologies
is sponsored by USENIX.

Engineering a backdoored bitcoin wallet

Adam Scott
Block, Inc.

Sean Andersen
Block, Inc.

Abstract

Here we describe a backdoored bitcoin hardware wallet. This wallet is a fully-functional hardware wallet, yet it implements an extra, evil functionality: the wallet owner unknowingly leaks the private seed to the attacker through a few valid bitcoin transactions. The seed is leaked exclusively through the ECDSA signatures. To steal funds, the attacker just needs to tap into the public blockchain. The attacker does not need to know (or control) any aspect of the wallet deployment (such as where in the world the wallet is, or who is using it). The backdoored wallet behavior is indistinguishable from the input-output behavior of a non-backdoored hardware wallet (it is impossible to discern non-backdoored signatures from backdoored ones, and backdoored signatures are as valid and just "work" as well as regular, non-backdoored ones). The backdoor does not need to be present at wallet initialization time; it can be implanted before or after key generation (this means the backdoor can be distributed as a firmware update, and is compatible with existing bitcoin wallets). We showcase the feasibility of the backdoored wallet by providing an end-to-end implementation on the bitcoin testnet network. We leak an entire 256-bit seed in 10 signatures, and only need modest computational resources to recover the seed.

Version: 2024-05-30

1 Introduction

Bitcoin, introduced in 2008 [Nak08], transacts in 2024 over 5 billion USD per day. Bitcoin is a distributed ledger that accumulates signed transactions conveying movement of funds. Cryptocurrencies like bitcoin are designed so that access to cryptographic keys directly controls the ability to move funds. In other words, in a cryptocurrency system, a security or cryptographic mistake normally results in the loss of funds.

Cryptocurrency users typically handle cryptographic keys in one of two ways: either they engage with a *custodian* to handle keys on the customer's behalf; or the users themselves store keys in specific-purpose *hardware wallets*.[1] Hardware wallets are essentially small-scale HSMs that at a minimum store (or derive) keys and sign transactions using public-key cryptography. Examples of hardware wallets are Trezor [Sat22] or Ledger [SAS22]. Hardware wallets typically use a key derivation strategy such as BIP32 [Wui12] to simplify storage requirements and overall key management complexity. In a deterministic key derivation strategy like BIP32, all private keys are deterministically derived from a master secret seed using a suitable key derivation function (like HMAC-SHA512 in the case of BIP32).

This paper focuses on cryptographic subversion of bitcoin wallets. We assume an attacker designs a backdoor and can inject custom firmware on the hardware wallet unit. The objective of the backdoor designer is to steal the wallet funds after the wallet is deployed. The custom wallet firmware contains a backdoored implementation of the signature scheme. This signature scheme is sound: it produces valid signatures. In addition, signatures themselves contain an extra, subliminal message embedded onto the signature values. This subliminal message is recoverable only to the backdoor designer. In our scenario, the subliminal message is the secret seed that generates all wallet private keys. Note that it is very easy for the attacker to obtain the signatures: since signatures are part of transactions and hence public information, the backdoor designer can easily recover the seed by monitoring the blockchain.

Previous work. Simmons introduced the problem of subliminal messages in signatures schemes and gave precise constructions back in the 1980s [Sim83, Sim84, Sim85]. Later Young and Yung generalized and coined the term "kleptography" to refer to backdoored cryptographic algorithms designed to steal information through covert channels [YY96, YY97a, YY04]. They also gave a very efficient kleptographic version of DSA that just requires two leaked

[1] Alternatively, the user can store the key directly on their personal device. This is considered to be riskier since general-purpose computers have a richer attack surface.

signatures [YY97b] assuming the signer is stateful. For formal definitions, see Ateniese et al. [AMV15, AMV20]. Cryptographic subversion is widespread: a notorious example of a (standardized) backdoor is Dual EC PRNG [SF07, BLN16].

Our contribution. This paper details the design and implementation of a flavor of Simmons backdoored DSA, heavily tailored towards an actual deployment onto a bitcoin hardware wallet. We focus on crafting a stealth backdoor that can be readily deployed in many different scenarios. We implement this backdoor end-to-end using the bitcoin testnet network and discuss optimizations and limitations. The backdoor designer can recover the whole seed with just 10 signatures and modest computational resources. We hope this example contributes to the development of more secure wallets and and ecosystems.

Applicability to other cryptocurrencies. This paper focuses on bitcoin, but the ideas easily transfer to other cryptocurrencies that use ECDSA with minimal modifications. Similarly, while this paper focuses on "personal" hardware wallets like Trezor or Ledger, the same observations carry to special-purpose, cold storage solutions.

1.1 Attacker model

We assume the adversary has control over the wallet code. Gaining control over a hardware wallet firmware can be attained by a variety of paths, including supply-chain attacks on any wallet software components, a compromise of the build system, a malicious insider with write access to source code or a compromise of the firwmare signing key.

On stealth backdoors. We note that when an attacker has achieved (essentially) remote-code execution in the hardware wallet firmware, there are many other exfiltration vectors available. For instance, a malicious wallet firmware could exfiltrate keys in unused fields in a bitcoin transaction, or resort to fancier physical exfiltration channels such as EM leakage or sound (if targeting a deep cold storage appliance). There is a plethora of work in this direction (sometimes called "covert channels") [KA98, VP09, LU02, GMME15, MSST05, GSE20]. However, these exfiltration channels often introduce new assumptions that significantly increase the cost of the attack: either by requiring a compromise of other system components (as needed if the exfiltration technique relies on introducing new messages) or some kind of physical proximity (as needed in all EM / sound leakage techniques). Our exfiltration technique works in a significantly more constrained setting (our attacker model is weaker) since we rely on less assumptions. We do not require compromise of additional system components nor require physical proximity (not even knowledge about the specific deployment). Thus, our techniques are more broadly applicable and can be universally deployed on bitcoin wallets.

2 Design space

Informally, we design a ECDSA signing algorithm featuring extra functionality: the signature values (r,s) convey extra information that allow the backdoor designer to extract the BIP32 seed. The basic idea traces back to Simmons, 40 years ago.

2.1 Backdoored wallet requirements

We present in this section a brief reminder of what makes a good cryptographic backdoor plus some particular properties of good backdoored wallets. This concept was first introduced by Young and Yung [YY96]. The following properties are desirable:

- R1 Backdoor is efficient, both for signing and leaking. On the one hand, this means there is no noticeable performance loss in the signer (signing is fast), and that only a handful of signatures are required to leak the secret (the subliminal channel has appropriate bandwidth). We assume the attacker can afford some moderate computation when recovering the secret from the signatures.

- R2 Backdoor is stealth: backdoored signatures should be computationally indistinguishable from non-backdoored ones. This ensures the backdoor is undetectable from its input/output behavior. Relatedly, the backdoor should be sound: leaked secrets are unrecoverable to anyone without the backdoor recovery seed.

- R3 Backdoor is suitable for deployment in a typical cryptocurrency scenario. This means that the backdoor system should tolerate loss of signatures (for example, a transaction is signed but never broadcasted) or reordering (in general, we cannot assume the order the signatures are generated is the same as transactions in the blockchain). The backdoor should be able to leak arbitrary data, not just the long term ECDSA key. Also, in typical deployments of hardware wallets, each signature is generated under a different long-term key (this is the case in BIP32). This means the backdoor should work when the long-term key changes from signature to signature. In addition, the backdoor should be stateless: in some systems, non-volatile storage may not be available (either as a security feature) or storing extra data in non-volatile storage may not be desirable (to lower detection probability, and simplify backdoor deployment).

2.2 Backdoor syntax

We first revisit the synatax of non-backdoored signature schemes and later augment it to construct backdoored signature schemes.

Digital signature. We adopt the usual syntax for a digital signature scheme, consisting of the following three algorithms:

- $\mathsf{Sig.KeyGen}() \rightarrow (\mathsf{pk}, \mathsf{sk})$ generates a public/private key pair.
- $\mathsf{Sig.Sign}(\mathsf{sk}, \mathsf{m}) \rightarrow \sigma$ emits a signature σ on a message m using private key sk.
- $\mathsf{Sig.Verify}(\mathsf{pk}, m, \sigma) \rightarrow \{0,1\}$ verifies signature σ on message m using public key pk.

For a definition of the ECDSA algorithm, see Appendix A.

Backdoored signatures. A backdoor signature scheme augments a signature scheme with two extra algorithms and modifies the signing algorithm:

- $\mathsf{BSig.SysParam}() \rightarrow (\mathsf{bp.sign}, \mathsf{bp.extract})$. Generates backdoor system parameters required for signing $\mathsf{bs.sign}$ and for recovery $\mathsf{bs.extract}$. The attacker typically runs this algorithm once to initialize the backdoor parameters.
- $\mathsf{BSig.Sign}(\mathsf{sk}, m, S, \mathsf{bp.sign}) \rightarrow \sigma$. Generates a backdoored signature σ on message m. The signature σ conveys the subliminal message S (potentially only partially) to be exfiltrated.
- $\mathsf{BSig.Recover}(\{\sigma_\mathsf{i}\}, \mathsf{bp.extract}) \rightarrow l$ recovers the subliminal message S from a set of signatures $\{\sigma_i\}$.

In practice, the signature scheme $\mathsf{BSig.Sign}$ may not fully leak the whole subliminal message S in a single signature, but only a part of it. To address this, the backdoor designer implements $\mathsf{BSig.Sign}$ to leak different parts of S on each invocation. Conversely, $\mathsf{BSig.Extract}$ takes multiple signatures $\{\sigma_i\}$ to reconstruct S. In other words, the backdoor designer may bake message fragmentation and reassembly into the backdoored algorithms.

3 Backdoor description / our construction

Basic idea overview. The backdoor designer wants to leak an arbitrary value S. For example, S could be the BIP32 seed that generates the whole wallet. The basic idea is to leak S in two steps. The first step is splitting S into several partial subliminal messages l_i. The second step is picking the ECDSA nonce to encode l_i into σ_i such that $\mathsf{BECDSA.Extract}$ can recover l_i from σ_i.

We write in Figure 1 the full description of the backdoored signature scheme BECDSA. In the next sections, we describe step by step the construction of BECDSA.

Backdoor description:

- $\mathsf{BECDSA.SysParam}() \rightarrow (\mathsf{bp.sign}, \mathsf{bp.extract})$. Generate an ECIES public/private key pair $\mathsf{esk}, \mathsf{epk}$. Generate a backdoor recovery secret $b \in_R \mathbf{Z}_q^*$. Return $\mathsf{bp.sign} \leftarrow (\mathsf{epk}, b)$ and $\mathsf{bp.extract} \leftarrow (\mathsf{epk}, b)$.
- $\mathsf{BECDSA.Sign}(\mathsf{sk}, m, S, \mathsf{bp.sign}) \rightarrow \sigma$. Signing consists of two parts. First we compute a subliminal message to leak $l = p(S, m, \mathsf{bp.sign})$. Then we generate a signature that leaks l: compute a nonce r from the message m, subliminal message l and backdoor recovery secret b as $r \leftarrow l \cdot h_1(m, b) \in \mathbf{Z}_q$. Call $\mathsf{ECDSA.Sign}(\mathsf{sk}, m; r)$ and emit the signature.
- $\mathsf{BECDSA.Recover}(\{\sigma_\mathsf{i}\}, \mathsf{bp.extract}) \rightarrow S$. Given a collection of signatures σ_i, extract each subliminal message $l_i \leftarrow \mathsf{Extract}(\mathsf{bp.recover}, \sigma_i)$ and then invert p using subliminal messages l_i to recover S.

Figure 1: BECDSA backdoor description.

Notation and parameters. We write h for a hash function, and add a subindex h_i when we want different hash functions to separate domains. We work with the prime order group $\mathbf{G}$ of elliptic curve points. In the bitcoin case, this is the secp256k1 curve.

3.1 Step 1: construction rationale

Here we detail the construction rationale for step 1, performing fragmentation and assembly.

Mapping S to l_i. We need a way to map the 256-bit S secret to several "short"[2] L-bit l_i. Simply assigning l_i to each L-bit chunk of S would work, but has important drawbacks. This raw method requires the receiver to receive all the subliminal messages l_i in order, and does not tolerate loss of l_i. In the context of bitcoin transactions, these two points can be hard to guarantee in practice since the signer may not broadcast every transaction (loss of l_i) or transactions may be broadcasted in a different order in which they were generated.

A more robust approach is to use the message m itself to "select" which linear combination of few bits from S are actually leaked. Since the message is public, $\mathsf{BECDSA.Recover}$ can invert this operation. More precisely, we use bit vector-matrix multiplication to "compress" S:

$$l_i \leftarrow S\mathbf{M} \qquad (1)$$

where S is seen as a 1×256 row vector of bits and $\mathbf{M}$ is a

[2] Typically, L is in the range of few dozen bits. This is a constraint coming from §3.2.

$256 \times L$ public matrix. The output l_i is L bits long. Intuitively, each l_i "picks" a "random" combination of bits from S.

The matrix $\mathbf{M}$ is spanned from the message m. Each message m generates a unique matrix $\mathbf{M}$. Since the message is public, the backdoor designer can easily reconstruct $\mathbf{M}$. For the concrete definition of $\mathbf{M}$ see Appendix B.

Recovering S from $\{l_i\}$. Recovering S from multiple l_i is very easy once enough l_i are recovered from the signatures σ_i. Each l_i adds L linear equations to a linear system of equations over $\mathsf{GF}(2)$. Solving for S recovers the seed.

Mapping properties. Using a map like Eq. (1) tolerates partial loss and reorder of some l_i. It also provides an entertaining feature: the backdoor designer gets an early progress report on how many bits from S are left to guess by computing the kernel dimension of the linear mapping. Note that it is not necessary to have a determined or over-determined system, but we only need to collect enough equations so that the last remaining bits can be bruteforced. We elaborate in §4.1.

3.2 Step 2: construction rationale

In what follows, r is the ECDSA secret nonce (also called short-term key) and the signature is the pair $\sigma := (c,s)$ where $c = f(g^r)$ (in our case, f just returns the x-coordinate of the curve point). For a complete description of ECDSA and notation, see §A.

Picking nonces. The basic working principle of backdooring ECDSA signatures is that the signer picks the ECDSA secret nonce r to convey the subliminal message l_i, a way that the backdoor designer can recover r and thus l_i. At the same time, the nonce r should still be unpredictable for someone that does not know the backdoor secret b. This ensures that the security of the ECDSA signatures is preserved. [3] An easy way to do this is by setting $r \leftarrow b \cdot l_i$.

Cross-stealing resistance. The basic method for cooking nonces $r \leftarrow b \cdot l_i$ makes a very fragile signature scheme. Since l_i is small, a collision between r_i can happen with large probability, leading to complete loss of security of the signing key (if the same key is used for different signatures[4]). To fix this, we diversify the backdoor recovery secret b on a per-message basis as $b_i \leftarrow h(b,m)$ where h is a suitable key derivation function and set $r \leftarrow l \cdot b_i$. This ensures the security of the signature scheme is preserved (no one but the backdoor designer can steal funds).

[3]Note that the "long-term" ECDSA key sk may change across invocations; thus, different signatures may correspond to different public keys. This happens in modern bitcoin wallets that are based on HD derivation [BIP32].

[4]While we cannot assume the same key sk is used for different messages, we cannot either discard this possibility, and the backdoor should yield a good signature scheme even if this is the case.

Recovering $\{l_i\}$ from $\{\sigma_i\}$. To recover l_i, the attacker computes the discrete logarithm of rG with respect of bG. This is relatively easy since l_i is small by construction. One straightforward procedure to solve this discrete logarithm is to just iterate over all possible 2^L values of l_i. More precisely, to recover l_i from a signature $\sigma_i = (c_i, s_i)$, we first unblind the first signature component $s = f(b_i l_i G)$ and compute the point $V_i \leftarrow b_i^{-1} b_i l_i G = l_i G$. From V_i, we extract l_i as $l_i \leftarrow \texttt{Extract}(V_i)$. This procedure simply iterates over all possible l_i until finding the value. We describe the computational optimizations to speed up this process below in §5
.

Discovery resistance. Assume the backdoor implementation gets leaked. This includes the secret b. Everyone who knows b can recover the leaked seed S. If this is a concern, then the backdoor implementation should leak a public-key encryption $E(S)$ of S instead of bare S. A good choice for the encryption functionality E is ECIES [Sho01] over secp256k1. This greatly simplifies the implementation as all the elliptic curve machinery is already in the signing implementation. The only drawback of this approach is ciphertext expansion: ECIES roughly doubles[5] the size of the subliminal leakage (512 bits), requiring more signatures to be leaked.

4 Discussion

4.1 How many signatures are needed to leak the full seed?

Tradeoff between L and number of signatures. There is a tension between the number of leaked bits per signature L and the number of required signatures to reconstruct the seed S. Obviously, we want to keep the number of required signatures as low as possible to leak the seed as soon as possible. That forces a high L, which in turns makes the recovery computationally expensive. The running time of $\mathsf{BSig.Recover}$ is exponential in L. We study this tradeoff in this section.

How much efficiency are we losing? The map from Eq. (1) is quite efficient: there is little redundancy across multiple l_i. Each l_i as generated per the mapping from Equation (1) leaks approximately L bits of S. This is because the rank of a random square matrix[6] over $\mathsf{GF}(2)$ is very close to its di-

[5]Pure trapdoor-based public key encryption do not yield a more efficient scheme. RSA encryption would add zero overhead, but obviously for the parameters in question RSA-256 would provide an unacceptable security level. Even state-of-the-art trapdoors with low overhead [DGH+19] still would be more expensive than KEM+DEM for such a short plaintext size (256-bit). For example, if the construction from [DGH+19] is asymptotically perfect (rate-1 ciphertext expansion), for 64-byte messages the ciphertext is 274-byte long [DGH+19, §6]. Another alternative is to use ECIES over a smaller curve.

[6]This naturally assumes the matrix $\mathbf{M}$ is constructed as per §B which can be considered uniform essentially random in $\{0,1\}$.

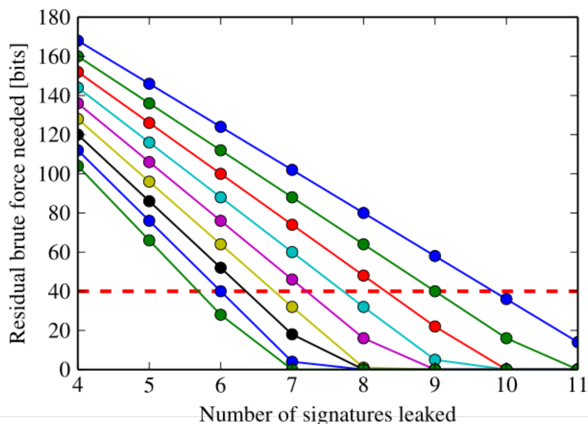

Figure 2: Remaining bruteforce effort needed to recover a unique S after solving the linear system from (1), for varying number of signatures leaked and bits per signature L. From top to bottom, each line corresponds to $L = 22, 24, \ldots, 38$. We plot a horizontal line at 40-bit effort level (representing a feasible bruteforce effort).

mension [Kol99, §3.2]. To get full rank with high probability, it suffices to add a few additional rows. This means that we need to leak around $256/L$ signatures to have a system of equations with almost unique solution for S.

Using less than $256/L$ signatures. Typically, we have access to an oracle that allows us to distinguish a correct guess for S from an incorrect one. (For example, when S is a seed, we can quickly tell if a guess for S is correct by checking if S unlocks some outputs.) This means that we do not need a fully determined system of equations to solve for S. When the system is not fully determined, the remaining few bits can be bruteforced by exhaustively generating all solutions for the system and checking each candidate. Note that generating solutions is a very efficient linear algebra operation (span the null space). This process only works when the remaining bits to be bruteforce is kept low (e.g. under 2^{40}).

Empirical results. In Figure 2 we empirically study how many signatures we need to leak the full secret S. As an example, we can see that if we allow $L = 35$ bits leaked per signature, after leaking 7 signatures there is only 11 bits left to bruteforce. This means that we can leak the whole 256-bit HD wallet seed with a handful signatures. Setting $L = 35$ bits per signature is feasible, as detailed below in §5.

4.2 Deterministic signatures

ECDSA signatures come in two broad flavors: randomized and derandomized (aka *deterministic signatures* [MNPV98, Por13]).[7] A backdoor signature scheme should mimic the existing wallet behavior to remain stealthy. Otherwise, the backdoored scheme is trivially distinguishable.

Deterministic backdoored signatures The scheme as described in Section 3 is deterministic, except for the ECIES encryption E for discovery resistance. To make ECIES deterministic, we can use a Encrypt-with-hash variant[8] from Bellare et al. [BBO07, §5.1]. This is possible in our case since the plaintext input to ECIES comes from a space with large min-entropy (a 256-bit seed S). This reduces to using a hash of the plaintext S as the randomness required for ECIES.

Randomized backdoored signatures. If the backdoor designer wishes to emulate a "randomized" version of ECDSA, they can randomize the backdoored nonce r by multiplying by a small integer v as $r \leftarrow v \cdot h(b,m) \cdot l_i$. This comes at a increased cost at recovery time (exponential in the bitsize of v). The random factor v should be large enough so that collisions are below a threshold backdoor detection probability. This is not a fully randomized ECDSA, since typically to be indistinguishable from a real random ECDSA signature, the value v would need to be very large (in the order of 128 bits). As a result, this backdoored ECDSA is not fully randomized, but may be useful to avoid light detection.

4.3 Recovery discussion

Variant: lighter recovery. To improve recovery speed, set $r \leftarrow b \cdot 2^{l_i}$ instead of $r \leftarrow b \cdot l_i$. This will speed up recovery, since it replaces elliptic curve point additions by point doublings, which are typically faster.

Identifying backdoored signatures. By design, there is no "in-protocol shortcut" to determine which signatures on the blockchain the recovery procedure should be applied to. This means that the backdoor designer should apply the recovery procedure to every signature, and discard those signatures for which the recovery procedure fails (i.e. does not yield a subliminal message l_i). Note that the backdoor designer could make use of additional side-channel information outside the raw ECDSA signature (like the way the transaction looks in

[7] Derandomized ECDSA signatures uses a deterministic process to generate the (pseudo-)random value k needed at signing time. They are prefered in practice since they are more resilient to imperfect randomness (at the cost of slightly increased computation). Many bitcoin wallets implement this strategy, usually in the form of RFC6979.

[8] This deterministic public-key encryption construction does not attain the usual standard level for encryption (semantic security) but in our specific case this is acceptable [BBO07].

vulnerable wallets) to speed up this process, but this is not necessary and orthogonal to our case.

Outsourcing recovery We note here that when using the discovery resistance feature from §3.2, it is possible to outsource the computationally expensive process in recovery to a different party. This party does not learn the content of S (only $E(S)$). Thus, this party, without access to bp.extract, cannot steal funds, nor correlate to a specific transaction. This can be useful to externalize this computationally expensive process. This party can also amortize its computational effort across different back door users, potentially by doing a heavy precomputation upfront and amortizing across different clients.

5 Recovery implementation

In this section, we focus on the `Extract` procedure, which is the most computationally demanding procedure from Recover. The $\texttt{Extract}(rP) \rightarrow r$ takes a curve point $Q = rP$ and outputs the discrete logarithm with respect to P, assuming r is bounded $0 < r < 2^L$.

Basic implementation. The basic implementation is just a linear search on l_i. $\texttt{Extract}(V_i)$ essentially loops sequentially over candidate l_i' till it hits $l_i'G = V_i$. The cost is 2^L elliptic curve additions and point comparisons, which is manageable when we keep L small. We can optimize this search at different levels.

Optimizations: baby-step giant-step. First, we can apply a classic time-vs-memory tradeoff (TMTO) by precomputing a table $T[j]$ storing M multiples of G, evenly spread over the search interval (from G to $2^L G$). This speeds up the search since `Extract` just needs to compute L/M additions $V_i + G, V_i + 2G, \ldots, V_i + \frac{L}{M}G$ and on each step check for inclusion on $T[j]$ to recover l_i. This is essentially baby-step giant-step algorithm to solve discrete logarithms.

Optimizations: point representation. Secondly, we can keep the points in Jacobian form (thus making point addition very fast) and perform the inclusion check on $T[j]$ after converting to affine representation. The speed-up comes from batching several points in this conversion and leveraging batched modular inversion.

Optimizations: compressed table. To lower memory requirements we can store compressed points in the table T (just the x-coordinate). This compression could be loosy (a short "fingerprint" of each point, such as some bits from the x-coordinate), at the cost of false positives (which can be filtered out easily).

Optimizations: parallelization. This search is amenable to parallelization at different levels. First, the search is embarrasing parallel on the search interval $1, \ldots, M$. Second, SIMD operations can speed-up the search by computing in parallel different chains of V_i. Note that in contrast with the usual context of elliptic-curve cryptography, the main objective in this search is to maximize *throughput* in point operations, not latency.

Implementation results. We wrote a prototype in Go featuring the TMTO optimization. This implementation tests about $2^{37.2}$ candidates l_i per second on a single core of a 2014 MacBook Pro with a table T holding 2^{22} points (compressed to 64 bits of the x-coordinate). The implementation is concise and takes around 30 lines of code. It is not particularly optimized for speed. It relies on `math.big` for multiprecision integers (field arithmetic is not optimized for secp256k1). For fast lookups, the table T is implemented as a hash map.

Real-time detection. In this section we see how quickly can an attacker leak a full 256-bit seed S if their computing power is the 2014 laptop from the section above. The fact that a single laptop can test about $2^{37.2}$ candidates per second means a single laptop can run `Extract` on real time on every transaction getting mined on the blockchain when each signature is leaking at most $L \leq 34$ bits. This rough estimate assumes the blockchain has a throughput of 8 transactions per second[9]. In turn, by looking up Figure 2, leaking $L = 34$ bits per signature means after just 7 signatures are leaked, there is enough information leaked to completely recover the seed S. (Leaking 7 signatures leaks about 238 bits, and the remaining 8 bits can be easily bruteforced).

Naturally, the estimations above are done with a single 2014 laptop as computing device. If the attacker has fancier hardware, they can use it to leak more bits per signature and accelerate the process, as they would need to leak less signatures.

Other techniques. It might be tempting to implement `Extract` based off generic techniques to solve the discrete logarithm problem in an interval. One such method is Pollard's lambda algorithm (also called Pollard's Kangaroo). This is left as future work.

6 Experiments

We implemented the backdoor end-to-end. This experimental backdoored wallet software runs on a laptop (rather than an actual hardware wallet) to make it easy to perform experiments. We implement the full backdoor except the "discovery

[9]This is a conservative estimation, the actual number is more between 3 and 7

resistance" feature (ECIES encryption) discussed in §3.2. The implementation is written in python for simplicity.

Results. We leak 19 bits per signature. The implementation signs 13 transactions. Signing overhead is negligible (an additional vector-matrix multiplication). The recovery process recovers the leaked arbitrary message in a matter of seconds. We set the leaked message to: 0x1234567890123456789012345678901234567890123456789012345678901234. We extract in total 247 bits from the signatures and the remaining 9 bits we just brute force by going through all the 2^9 solutions to the linear system of equations.

Transaction hashes. We put a chain of 13 transactions on signet (a Bitcoin staging network) starting with the transaction f019bd3461309ae48c48a9cee5edaefb8ff4ef4c921fbd9d43377ff64162d77b. The full list of transactions can be found in Appendix C and the example output of the recovery tool can be found in Appendix D.

Generalizations. Whilst these concrete transaction chain uses the same private key for every transaction, this is coincidental (to make the implementation easier) and not essential. Therefore, the backdoor is compatible with BIP32 wallets. Also, even if these concrete transactions are chained, this is not a requirement. The different backdoored signatures could come from unrelated, unlinked transactions.

7 Detection, deployment

7.1 Distinguishing backdoored signatures

We elaborate here on the requirement R2 from §2.1.

Unknown-key scenario. For an observer that does not know the secret key material (but only observes the black-box, input/output behavior of the wallet), backdoored signatures are indistinguishable from regular ones. This follows from the PRF security of the hash function $h(b,m)$.

Known-key scenario. For an observer that knows secret key material (or can choose the key), the backdoored scheme is trivially distinguishable. For example, the backdoored scheme will not pass test vectors. This observer can take the reference test vectors from a known-good implementation (or from the signature algorithm specification, such as RFC6979). Note that a backdoor could easily hide itself (by computing non-backdoored signatures) whenever it detects from its environment that it is running in test mode (for example, the backdoor could detect if it is being fed test vector inputs from a publication) or is running in a developer machine, or in the software build pipeline, or is using a test key, or the key was not generated internally at random inside the signer device.

7.2 Deployment aspects

Build pipeline. An attacker can plant the backdoor by compromising the software build pipeline. These systems are typically operated by different teams and could become an easy target.

Stealing firmware signing keys. Alternatively, this backdoor could also be planted by stealing the firmware signing keys. Conversely, note that the firmware writer has a *lifelong liability* to keep this signing key safe.

Evil maid. The evil maid is a particularly attractive vector for introducing this backdoor. Say you buy a Trezor or Ledger from Amazon. Replacing the whole hardware wallet with an evil, backdoored one is an option to deploy this backdoor. Ironically, the fact the code is open-source makes this process extremely easy. (Maybe by sending the wallet back to Amazon through the RMA process after having injected the backdoor.)

Implementing the backdoor at other abstraction levels. In this paper, we assume we implemented the backdoor at the application firmware level. The backdoor could be implemented at other levels: at the OS level (detecting whenever the ECDSA nonce is generated), or at a hardware level (for example by tampering with the RNG peripheral on nonce generation.)

Lowering detectability. The retrieval strategy is out of scope of this document. One possibility is to plant this backdoor in many wallets, but steal funds only from a few well-funded wallets. This can be used to lower the suspicion on a systemic breach like a firmware compromise. For example, one could only siphon out the top 0.1% of the wallets.

Multi-user setting. The backdoor description in §3 assumes we want to leak a secret from a single wallet. We can extend this to the multi-user setting (multiple wallets) in several ways. We can diversify the backdoor recovery secret b on a per-wallet basis. This is a clean approach on the wallet side; the recovery effort increases linearly with the number of backdoored wallets. Alternatively, the w-th wallet can leak first a single, short subliminal message l_1^w using a global backdoor recovery secret b. This l_1^w encodes a "session" backdoor recovery secret $b_w = \texttt{KDF}(b, l_1^w)$. The w-th wallet uses this "session" recovery secret b_w for subsequent subliminal messages. This makes the complexity of recovery substantially smaller.

7.3 Comparison

Our construction relies on subliminal messages in ECDSA. Simmons already provided in the 1980s several constructions

for this [Sim83, Sim84, Sim85] that relied on manipulating the DSA secret nonces. Our construction adds a layer on top for message fragmentation.

The early construction of Young and Yung [YY97b] is very efficient, requiring only two signatures. However, it needs to set the same key for both signatures, and the signer must be stateful. In an actual hardware wallet, statefulness may be hard to guarantee since this requires write capabilities to non-volatile storage (which may be not even present).

The construction of Ateniese et.ãl [AMV15] is tangentially related to ours. However, the running time for the backdoored signer is exponential in the number of bits leaked, which makes the backdoor easy to detect by measuring execution time.

8 Mitigations

Split trust: multisignatures. Many protocols (including Bitcoin) accept multisignatures natively. This consideration can be taken at design time to generate a 2-of-2 wallet between the host and the hardware wallet. This technique can be used to avoid the consequences of a backdoored signature, but comes at the price of longer transaction (hence more expensive) and more complexity.

As with any technique that relies on splitting trust, diversity and heterogeneity are critical to actually gain security. In this case, if both the software running in the host *and* the firmware running in the hardware wallet are developed by the same teams, the cost of mounting an attack against both is not much higher.

Split trust: firewalled signatures. As noted by Dauterman et al [DCM+19], this problem can be solved with cryptographic reverse firewalls [MS15]. This is a general technique that protects against cryptographic subversion and assumes some system parts (the reverse firewall) are trusted and behave correctly. Cryptographic reverse firewalls can be built from zero-knowledge techniques, but the performance is typically much worse than custom designs such as [DCM+19].

Split trust: multi-party computation. Firewalled signatures can be implemented also with multi-party computation. There are a myriad of threshold ECDSA designs that could be used [Lin17, GG18, CCL+19, MPS19, CGG+20].

On the more practical side, Dauterman et al [DCM+19] design a lightweight 2-party protocol for firewalled ECDSA signatures between a signer and a firewall. By construction, the signer cannot exfiltrate any message via bits of the signature. At first sight, it is easy to fall into this circular reasoning: what does this buy us if we anyways have to trust the firewall? This construction is appealing since the firewall itself does not require to store any secret key material (thus making it easy and cheap to manufacture with commercial, off-the-shelf parts, and hence trust).

We see an opportunity in standardizing the protocol the signer and the firewall (potentially, multiple firewalls) so that interoperability between different manufacturers for the firewall and signer is possible. Diversity here is beneficial for security.

At run time: attestation. One way to mitigate evil-maid style attacks is by using attestation: the hardware wallet could prove its authenticity to the host before the host trusts the hardware wallet.[10] This requires setting some kind of PKI between the hardware wallet manufacturer and host software; and heavy modifications to the hardware wallet (quote generation functionality and provisioning secrets or certificates in the hardware wallet.)

General supply-chain mitigations. A backdoored wallet is a hardware and software supply chain problem, hence, generic mitigations against supply chain threats apply. These are not specific to the problem of a backdoored wallet, but apply to every security-critical hardware or software product. Without being exhaustive, generic mitigations like code audit, code signing, build system hardening, artifact store hardening, reproducible builds, artifact signing and secure boot will help.

Strengthening firmware signing. A take away is that we need to make firmware signing more robust. Firmware signing keys protect too much value, so it is convenient to diffuse this pressure. One way for example is by using multiple firmware signing keys, each owned by a different party that performs independent authorization of the signing action. This is essentially a straight multi-signature scheme. One can use more complex multi-signature schemes like MuSig2 [NRS21]. Potentially some of the firmware signing keys could be stored offline with a tight, verifiable log of key utilization.

At validation time: known-answer tests. A mitigation strategy could be cross-validation with a known-good implementation. As noted in §7.2, the backdoor could recognize the test vector inputs and react accordingly to hide the backdoor. This can be mitigated by using random, unpredictable inputs. In addition, the backdoor could sense the environment that is currently running on, and only get activated in a production environment, while the debug/development builds hide the backdoor behavior.

Applicability to other wallets. We focus here on hardware wallets, but the same principles could be applied to hot wallets or purpose-specific cold storage appliances.

[10] We need some mental gymnastics when laying out the threat model here since the raison d'être of hardware wallets is the host is untrusted.

9 Lessons learned

We collect here lessons we learned that could be useful in the threat modeling process. The lessons here can assist the security architect in gauging the risk level when developing Bitcoin wallets.

Firmware signing key: lifetime responsibility. The technique presented in this paper can be used to turn an existing, uncompromised wallet into a backdoored one. This means the wallet designer has a lifetime responsibility of safeguarding the firmware signing key (provided the wallet has some kind of firmware update mechanism).

Firmware signing key: value. The monetary value of the firmware signing key can be roughly estimated as the sum of the wallet balances where the signed firmware runs. This is typically much larger than the firmware signing key of consumer electronics. Thus, significant more resources should go to protect this key.

Build system: value. The previous observation carries also to the build system: the cost of sneaking a backdoor anywhere in the supply chain should be commensurate to the reward of backdoor wallets. Otherwise, there's an opportunity for an attacker to make a profit.

The previous two points make clear the need for making firmware signing robust. We described mitigations in §8.

Eliminating exfiltration channels is not enough. A common good practice is to reduce or eliminate the "free slots" to exfiltrate data from the signer. This could take the shape of exfiltrating data through unused data fields (in case the API allows that), or by using representations that are not deterministic (i.e. admit several different representations for the same input). Whilst this is generally a good idea, it is not enough, since just the signature is enough to exfiltrate seeds in a backdoored wallet.

Physical exfiltration ranks lower in priority. For the security architect designing a Bitcoin wallet, physical exfiltration concerns rank lower than supply-chain security, since attacks using physical exfiltration are strictly harder to mount than supply-chain ones.

Compromise of the signer is enough. A relevant metric in assessing the attack difficulty or chances to get detected is how many different components need to be compromised. In this case, we do not need to compromise any other system upstream of the signer. (The attack works if the host computer that the hardware wallet is connected to remains uncompromised.)

Air-gap may give false sense of security. The backdoored wallet described here works also in "air-gapped" systems. By definition, in an air-gapped wallet the signature will always need to cross the air gap, and hence air-gapping is not enough to prevent this attack (even if it can help to reduce the attack surface.)

RMA process can be a can of worms. The reverse order fulfilling system is also a good opportunity to inject backdoors: a malicious user may return a wallet claiming it does not work properly, in the hope that the wallet gets later sold to another customer as a refurbished device. The wallet manufacturer is thus forced to inspect the wallets for backdoors if they want to sell later refurbished devices. This is a very hard problem.

10 Conclusion

In this paper, we engineered a bitcoin backdoored wallet. The backdoor is very efficient and can leak a full seed in about a dozen signatures. We demonstrated the feasibility of our approach by implementing an end-to-end demo. We hope some of our observations help the development of hardware wallets.

Future work. This backdoor is efficient, but could scale better in the number of injected backdoors. The effort required to recover exfiltrated seeds is linear in the number of injected backdoors and this could be improved.

Ethical considerations. The observations in this paper could be used to cause harm. We have performed our experiments in a non-production network using mock values. We believe the recommendations in this paper for mitigating backdoored wallets outweigh the potential harm.

A ECDSA definition

We adopt the same notation and exposition as [FKP16]. We have a prime order group $\mathbf{G}$ of order q generated by $g \in \mathbf{G}$, $H : \{0,1\}^* \to \mathbf{Z}_q$ a hash function, and f a "conversion function" $f : \mathbf{G} \to \mathbf{Z}_q$ (in ECDSA f typically returns the x-coordinate of the input curve point). We define ECDSA as the following three algorithms:

- $\mathsf{ECDSA.KeyGen}() \to (\mathsf{pk}, \mathsf{sk})$. Sample $x \in_R \mathbf{Z}_q$ at uniform random. Set x as the private key and $X = g^x \in \mathbf{G}$ as the public key.
- $\mathsf{ECDSA.Sign}(\mathsf{sk}, m) \to \sigma$. Pick a random nonce $r \in_R \mathbf{Z}_q^*$. Compute $c \leftarrow f(g^r)$ and $s \leftarrow r^{-1} \cdot (H(m) + c \cdot x)$. Output $\sigma \leftarrow (c, s) \in \mathbf{Z}_q^2$. When the caller picks the random nonce r, we write $\mathsf{ECDSA.Sign}(\mathsf{sk}, m; r)$.

```
f019bd3461309ae48c48a9cee5edaefb8ff4ef4c921fbd9d43377ff64162d77b
32d31db85be4f21c0d1761f4871ec1dde6cd9b2bede0b83aa850262fb401ba32
a2a40cbf8e85cf315ef9e83efc148b67103e92feb6d52403bee62e0cdb2aede8
8d42358580badccd94407d279225829ffaf2215dcc508877bc12e3ae36967673
1db369cc29eee5330265563460ba939c19c0bc61ae7dd155b98329b482fc382b
5a63c79eca5de1fd901be18f98d41ce1a1e4dc5acc2fa0bb5a440b6c0ee4ca4a
71cf01fdfe9f3566a9e07b9535fff9d173f51a7e7f68dff253e9fafba1930a58
83b0d44575af054e035c42bb4cef8d1e23440c490d383fa6c0e4bc32b14888cb
303a9ab5e5763e66b2cee395de05db15648888b48ac7f19dffc5b85496804a50
55929d6b5f7c1973f1f44dce5bc2b687c6e667cd26edf56a968b03eac0abf9d7
a8563f37969ac19ce41137bac194f69169ba82670c96cdfad25405f720ddf100
350b8ce7186015283cd09fbcb2cbbd546bc824c7d7b8518fd95e5e7af9c4be13
a0eb336a63e3eb246dfd89f9eb7d128f54bc980fa5443b2fdb3c828e5760f9db
```

Figure 3: Transaction hashes for 13 transactions that leak a backdoor

```
>> recovering 19 bits per sig
   recovered 0x050942 from sig 0
   extracted 0x02af5b from sig 1
   extracted 0x05d30b from sig 2
   extracted 0x013ab6 from sig 3
   extracted 0x03d2d2 from sig 4
   extracted 0x043926 from sig 5
   extracted 0x05e140 from sig 6
   extracted 0x07e246 from sig 7
   extracted 0x0742ac from sig 8
   extracted 0x03ea4e from sig 9
   extracted 0x0185a0 from sig 10
   extracted 0x05915a from sig 11
   extracted 0x01af06 from sig 12
>> exhaustive search
   bits to guess: 9
   brute force successful!
    number_signatures=13
    leaked_bits_per_sig=19
    dim(ker(A))=9
   recovered=1234567890123456789012345678901234567890
             123456789012345678901234
```

Figure 4: Recovery tool output

- $\mathsf{ECDSA.Verify}(\mathsf{pk}, m, \sigma') \to \{0,1\}$. Write X for the public verification key. The purported signature $\sigma = (c, s)$ is valid if and only if $f(R')$ is equal to c, where $R' = (g^{H(m)} X^c)^{1/s} \in \mathbf{G}$.

B Definition of M

The matrix $\mathbf{M}$ is a $256 \times L$ random-looking matrix spanned by the public message m. We compute L different 256-bit hashes $h_1(m), \ldots, h_L(m)$ of m and stack them to construct $\mathbf{M}$. (For example, $h_i(m) := h(\mathtt{i}||\mathtt{M\text{-}matrix}||m)$ This ensures the matrix has balanced statistics and we can model it as uniformly random bits for the purposes of Section 4.1.

C Transaction hashes

In Figure 3 we write hashes for 13 transactions. We leak 19 bits per transaction.

D Recovery output

In Figure 4 we put an exemplary output of the recovery process. The tool recovers the leaked seed from 13 signatures after trying 2^9 candidates.

References

[AMV15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 364–375. ACM, 2015.

[AMV20] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signatures: Definitions, constructions and applications. *Theor. Comput. Sci.*, 820:91–122, 2020.

[BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 535–552. Springer, 2007.

[BLN16] Daniel J. Bernstein, Tanja Lange, and Ruben Niederhagen. Dual EC: A standardized back door. In Peter Y. A. Ryan, David Naccache, and Jean-Jacques Quisquater, editors, *The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, volume 9100 of *Lecture Notes in Computer Science*, pages 256–281. Springer, 2016.

[CCL+19] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 191–221. Springer, 2019.

[CGG+20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Jay Ligatti,

Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1769–1787. ACM, 2020.

[DCM+19] Emma Dauterman, Henry Corrigan-Gibbs, David Mazières, Dan Boneh, and Dominic Rizzo. True2f: Backdoor-resistant authentication tokens. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 398–416. IEEE, 2019.

[DGH+19] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Kevin Liu, and Giulio Malavolta. Rate-1 trapdoor functions from the diffie-hellman problem. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 585–606. Springer, 2019.

[FKP16] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the provable security of (EC)DSA signatures. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1651–1662. ACM, 2016.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1179–1194. ACM, 2018.

[GMME15] Mordechai Guri, Matan Monitz, Yisroel Mirski, and Yuval Elovici. Bitwhisper: Covert signaling channel between air-gapped computers using thermal manipulations. In Cédric Fournet, Michael W. Hicks, and Luca Viganò, editors, *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 276–289. IEEE Computer Society, 2015.

[GSE20] Mordechai Guri, Yosef A. Solewicz, and Yuval Elovici. Fansmitter: Acoustic data exfiltration from air-gapped computers via fans noise. *Comput. Secur.*, 91:101721, 2020.

[KA98] Markus G. Kuhn and Ross J. Anderson. Soft tempest: Hidden data transmission using electromagnetic emanations. In David Aucsmith, editor, *Information Hiding, Second International Workshop, Portland, Oregon, USA, April 14-17, 1998, Proceedings*, volume 1525 of *Lecture Notes in Computer Science*, pages 124–142. Springer, 1998.

[Kol99] Valentin F. Kolchin. *Random graphs*, volume 53 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1999.

[Lin17] Yehuda Lindell. Fast secure two-party ECDSA signing. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 613–644. Springer, 2017.

[LU02] Joe Loughry and David A. Umphress. Information leakage from optical emanations. *ACM Trans. Inf. Syst. Secur.*, 5(3):262–289, 2002.

[MNPV98] David M'Raïhi, David Naccache, David Pointcheval, and Serge Vaudenay. Computational alternatives to random number generators. In Stafford E. Tavares and Henk Meijer, editors, *Selected Areas in Cryptography '98, SAC'98, Kingston, Ontario, Canada, August 17-18, 1998, Proceedings*, volume 1556 of *Lecture Notes in Computer Science*, pages 72–80. Springer, 1998.

[MPS19] Antonio Marcedone, Rafael Pass, and Abhi Shelat. Minimizing trust in hardware wallets with two factor signatures. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11598 of *Lecture Notes in Computer Science*, pages 407–425. Springer, 2019.

[MS15] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part*

II, volume 9057 of *Lecture Notes in Computer Science*, pages 657–686. Springer, 2015.

[MSST05] Anil Madhavapeddy, Richard Sharp, David J. Scott, and Alastair Tse. Audio networking: the forgotten wireless technology. *IEEE Pervasive Comput.*, 4(3):55–60, 2005.

[Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. `https://bitcoin.org/bitcoin.pdf`, 2008.

[NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. Musig2: Simple two-round schnorr multi-signatures. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 189–221. Springer, 2021.

[Por13] Thomas Pornin. Deterministic usage of the digital signature algorithm (DSA) and elliptic curve digital signature algorithm (ECDSA). *RFC*, 6979:1–79, 2013.

[SAS22] Ledger SAS. . `http://www.ledger.com/`, 2022.

[Sat22] SatoshiLabs. Trezor. `http://trezor.io/`, 2022.

[SF07] D. Shumow and N. Ferguson. On the possibility of a back door in the NIST SP800-90 dual EC PRNG. `rump2007.cr.yp.to/15-shumow.pdf`, 2007.

[Sho01] Victor Shoup. A proposal for an ISO standard for public key encryption. *IACR Cryptol. ePrint Arch.*, page 112, 2001.

[Sim83] Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In David Chaum, editor, *Advances in Cryptology, Proceedings of CRYPTO '83, Santa Barbara, California, USA, August 21-24, 1983*, pages 51–67. Plenum Press, New York, 1983.

[Sim84] Gustavus J. Simmons. The subliminal channel and digital signature. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *Advances in Cryptology: Proceedings of EUROCRYPT 84, A Workshop on the Theory and Application of of Cryptographic Techniques, Paris, France, April 9-11, 1984, Proceedings*, volume 209 of *Lecture Notes in Computer Science*, pages 364–378. Springer, 1984.

[Sim85] Gustavus J. Simmons. A secure subliminal channel (?). In Hugh C. Williams, editor, *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 33–41. Springer, 1985.

[VP09] Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In Fabian Monrose, editor, *18th USENIX Security Symposium, Montreal, Canada, August 10-14, 2009, Proceedings*, pages 1–16. USENIX Association, 2009.

[Wui12] Pieter Wuille. Hierarchical deterministic wallets. *Bitcoin Improvement Proposal (BIP)*, 32, 2012.

[YY96] Adam L. Young and Moti Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 1996.

[YY97a] Adam L. Young and Moti Yung. Kleptography: Using cryptography against cryptography. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1997.

[YY97b] Adam L. Young and Moti Yung. The prevalence of kleptographic attacks on discrete-log based cryptosystems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 264–276. Springer, 1997.

[YY04] Adam L. Young and Moti Yung. *Malicious cryptography - exposing cryptovirology*. Wiley, 2004.