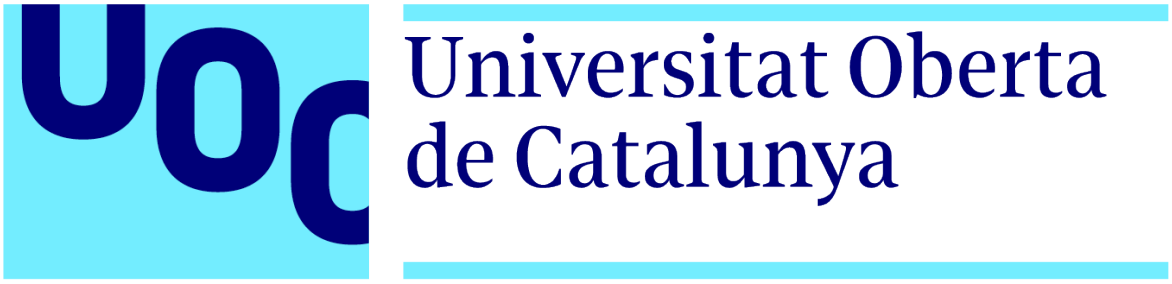


Analysis of Covert-Nonce Channel Attacks on Bitcoin Hardware Wallets

Domènec Madrid Hernández

Tutora del TFM: Cristina Pérez Solà

Màster Universitari en Ciberseguretat i Privadesa

Universitat Oberta de Catalunya

03 Juny 2025

Resum

Quan parlem de hardware wallets en bitcoin, suposem que pel fet de trobar-se completament aïllades d'Internet, ser específicament dissenyades per assolir la màxima seguretat i capaces d'operar sense una connexió directa amb l'ordinador, són considerables com una fortalesa impenetrable, res no podria estar més lluny de la realitat. Tot i ser molt més segurs que la majoria de les solucions, aquests dispositius continuen sent vulnerables a vectors d'atac altament sofisticats i capaços d'extraure la informació més sensible que pot contenir una bitlletera en bitcoin: la llavor. Això pot comportar la pèrdua total dels fons associats a les claus privades, tot plegat de manera completament invisible per a l'usuari i de forma totalment remota.

Aquest treball de final de màster té com a objectiu explorar els diferents atacs de canal encobert que exploten els valors aleatoris de les signatures criptogràfiques probabilístiques utilitzades en bitcoin. D'altra banda, s'hi analitza com es poden dur a terme aquest tipus d'atacs, si existeixen contramesures actuals per mitigar-los i, finalment, si les solucions més avançades en seguretat, com ara les hardware wallets, ja les implementen.

Abstract

In the context of Bitcoin hardware wallets, is often assumed that due to its complete isolation from the Internet, specifically designed to achieve maximum security and capable of operating without a direct connection to a computer, they are an impenetrable fortress. Nothing could be further from the truth. Although significantly more secure than most other solutions, these devices remain vulnerable to highly sophisticated attack vectors capable of extracting the most sensitive piece of information that a Bitcoin wallet can hold: the seed. This can lead to the total loss of the funds associated with the private keys, all in a way that is completely invisible to the user and entirely remote.

This master's thesis aims to explore the different covert channel attacks that exploit the random values used in probabilistic cryptographic signatures in Bitcoin. It also analyzes how such attacks can be carried out, if there are existing countermeasures to mitigate them, and, finally, whether the most advanced security solutions, such as hardware wallets, already implement these protections.

Keywords: *Covert nonce channel attack, Hardware wallet security, Cryptographic signature vulnerabilities, ECDSA, Schnorr, Bitcoin*

Contents

1 Introduction

The security of cryptocurrencies is of paramount importance. In decentralized digital currencies such as Bitcoin, private keys are the fundamental mechanism used to authenticate users and authorize transactions. These keys grant control over funds, meaning that the person who possesses the private key can sign and transfer assets. Therefore, secure management of private keys and isolation from the internet are crucial to ensuring the safety of digital assets.

As the cryptocurrency market has grown, sophisticated cryptographic devices have been developed to secure private keys. Some of them, known as hardware wallets, provide high-level security by keeping private keys isolated from online environments, making it extremely difficult for attackers to compromise them.

However, as with the Trojan horse, these devices can still be vulnerable if their firmware has been tampered with before or after purchase. Such modifications may occur during manufacturing, distribution, or even during possession of the user [22]. Attackers can introduce undetectable malicious firmware modifications that allow them to carry out illicit activities, making it nearly impossible for the user to identify the breach.

This research focuses on a specific class of attacks targeting Bitcoin hardware wallets, known as exfiltration attacks or covert nonce channel attacks [35]. In these attacks on a blockchain ecosystem, a malicious actor is already embedded within the user's hardware. The primary goal of this attack is to exfiltrate the wallet's seed, its core cryptographic secret, through the transaction signing process. This is achieved by manipulating the nonce used in the signature process. By subtly encoding information in the nonce, an attacker could gradually leak the private seed, allowing him to recover the user's private key solely by observing transactions recorded on the blockchain.

1.1 Justification

As the world of cryptocurrencies grows, more and more people are starting to focus their attention on it. The fact that the price of Bitcoin has reached more than 100.000$ [17] suggests that this digital asset could become a mainstream currency in the near future. For this reason, it is crucial that security is top-notch and, at the same time, easy to use for the average user.

Although self-custody remains a complex aspect of the Bitcoin world, those who own large amounts of Bitcoin use it to ensure that their assets are not stolen by unauthorized persons [7].

In this context, the security of hardware wallets, the flagship device for Bitcoin self-custody, is a key issue. It is imperative to manage vulnerabilities such as covert nonce channel attacks and other types of attacks that could compromise all funds in just a few steps.

This study aims to contribute to the field by analyzing current measures against such attacks and reviewing if a better solution could be implemented. Furthermore, this research aligns with the ninth goal of the Sustainable Development Goals of the United Nations, which promotes innovation and resilient infrastructures. In this study, we specifically focus on digital security as applied to the cryptocurrency ecosystem.

1.2 Personal Motivation

Since I had taken the Blockchain and Cryptocurrencies course during my degree, my interest in this field has been growing. After securing a job as a researcher at UAB in the Discrypt research group, I have managed to align my hobby and interest with my work.

After working for almost a year on similar projects, this project emerged from tests with signatures on various hardware wallets. In the subsequent analysis, we observed that when signing the same transaction, several wallets generated different signatures. This suggests that not all of them are implementing RFC 6979, which defines a deterministic nonce for signing in Bitcoin.

This finding led us to explore various lines of research, concluding that one of the most relevant was the study of how these signatures are managed and what security implications they might have. In particular, we focused on the possibility that a modified firmware could lead to covert channel attacks, where an attacker could extract sensitive information by analyzing variations in the signatures.

1.3 Research Goals

The main objective of this study is to analyze various attack vectors associated with covert channels and to evaluate the effectiveness of existing countermeasures, while exploring potential improvements to enhance current security protocols.

1.3.1 Hypothesis

If an attacker manages to modify the firmware of a hardware wallet to manipulate the nonce used in a probabilistic signature scheme, they could covertly exfiltrate information, allowing them to reconstruct the user's seed by analyzing transactions on the blockchain.

1.3.2 Objectives

The following objectives define the scope of this research. Since the primary focus is theoretical, the study will focus on reviewing and analyzing existing literature on these attacks and their possible solutions. The implementation of a new protocol is considered an optional objective, depending on the availability of time for its development.

- **O1: Analyze the vulnerability of covert nonce channel attacks in hardware wallets**
  - *Study how an attacker can manipulate the firmware to exfiltrate the private key through the transaction signing process.*
  - *Investigate whether there is a way to detect abnormal patterns to identify compromised wallets.*
- **O2: Evaluate the implementation of RFC 6979 in different hardware wallets**
  - *Verify whether the analyzed wallets use a deterministic nonce according to RFC 6979 or other implementations.*
  - *Investigate how a nondeterministic implementation can facilitate covert nonce channel attacks.*
  - *Observe whether an alternative or improved implementation is used if RFC 6979 is not applied.*
  - *Identify which hardware wallet models are more susceptible to this type of attack.*
- **O3: Validate the feasibility of the attack in a controlled environment**.
  - *Conduct proof-of-concept tests to demonstrate the feasibility of an exfiltration attack in a custom test network.*
  - *Measure the time and resources required to extract a private key using this method.*
- **O4: Analyze the impact of these attacks on the security of the Bitcoin ecosystem**
  - *Study how a successful attack could affect both users and trust in hardware wallets.*
  - *Assess the real-world risk of this type of attack being exploited.*
- **O5 (Optional): Explore and propose a protocol to improve the current anti-exfiltration system**

1.4 Methodology

The study of relatively new technologies implies that the available information comes largely from sources such as personal blogs, forums, or repositories like GitHub, making the search for academic papers more complex. That is why this project will be based on a review of the state of the art, considering not only academic papers but also technical blogs, forum discussions, answers on platforms like Stack Exchange and other relevant sources. Although its scarcity of information, specific attention will be paid to the credibility of the sources checked, filtering resources by reputation or known authors. Beyond merely analyzing existing research, this review will also lay the groundwork for developing improvements to the protocol or improving some open-source wallets.

For this reason, the adopted methodology will follow the design and creation approach, as proposed by Briony J. Oates in her book *Researching Information Systems and Computing* [11]. This methodology is well suited to the process of analysis, design, and implementation that this master's thesis seeks to achieve.

As Oates points out, developing a new protocol in an environment with limited information does not allow for a classical development model. Therefore, an iterative prototyping process will be used, following **experiments** methodology in which appropriate tests will be carried out to confirm that our hypothesis is correct and to improve our solution iteration by iteration. This approach will allow us to expand our understanding of the problem and gradually refine the solution until an optimal implementation is achieved.

Although a large part of this work will be theoretical, there is also a practical component that relies on both software and hardware tools. To establish a fully controlled testing environment, a private Bitcoin network will be launched using Docker containers running Bitcoin Core, each container operating as a node in the network ensuring complete isolation. Transaction construction will be performed using the Sparrow Wallet connected to these test nodes. Signature creation will rely on hardware wallets, which will function either air-gapped or physically connected; their selection will follow specific criteria to ensure compatibility and utility. Each of these hardware wallets will be integrated into the network and validated for compatibility, then subjected to a detailed analysis of nonce behavior to determine whether they adhere to standards or employ alternative methods. Lastly, the resulting data will inform the identification of potential covert-nonce channels and guide proposals for improved algorithms that address issues related to the probabilistic nature of digital signature schemes in Bitcoin, while also supporting best practices for hardware wallet integration.

1.5 Task Breakdown

1.5.1 Process Phases

1. **Analysis of Existing Strategies**
   (a) The different strategies implemented to prevent exfiltration attacks will be reviewed and their effectiveness evaluated.
   (b) Current methodologies will be documented and contrasted.
2. **Selection and Evaluation of Hardware Wallets**
   (a) A preliminary selection of hardware wallets to be analyzed will be made based on criteria such as popularity, technical implementation, open source nature, and availability.
   (b) It will be verified whether these wallets comply with the **RFC 6979** standard for deterministic nonce generation or use other implementations.
3. **Implementation of a Realistic Test Environment**
   (a) A parallel mainnet will be created to allow transactions to be carried out securely and to observe the results of each implementation.
   (b) The environment will be designed to replicate, as closely as possible, real-world conditions for hardware wallet usage.
4. **Dynamic Signature Analysis**
   (a) Experimental tests will be conducted to observe the signatures generated by the selected hardware wallets.
   (b) The obtained values will be compared to detect anomalous patterns or inconsistencies in nonce generation.
5. **Implementation of the Attack in a Controlled Environment**
   (a) The firmware of the hardware wallets will be modified to include nonce manipulation mechanisms, and it will be verified whether it is possible to exfiltrate the private key without noticing.
   (b) The time and resources required to carry out the attack will be measured, and its feasibility in a real-world scenario will be analyzed.

6. **Validation and Proposal of Mitigation Protocols**

   (a) If vulnerabilities or areas for improvement are identified, a new protocol will be proposed to mitigate the risk of private key exfiltration through nonce manipulation.

7. **Creation of a Review Article**

   (a) A comprehensive report will be prepared summarizing the findings of the analysis and experiments.

   (b) The document will be structured to facilitate peer review and potential publication, ensuring clarity, reproducibility, and relevance to the security research community.

1.5.2 Project Planning

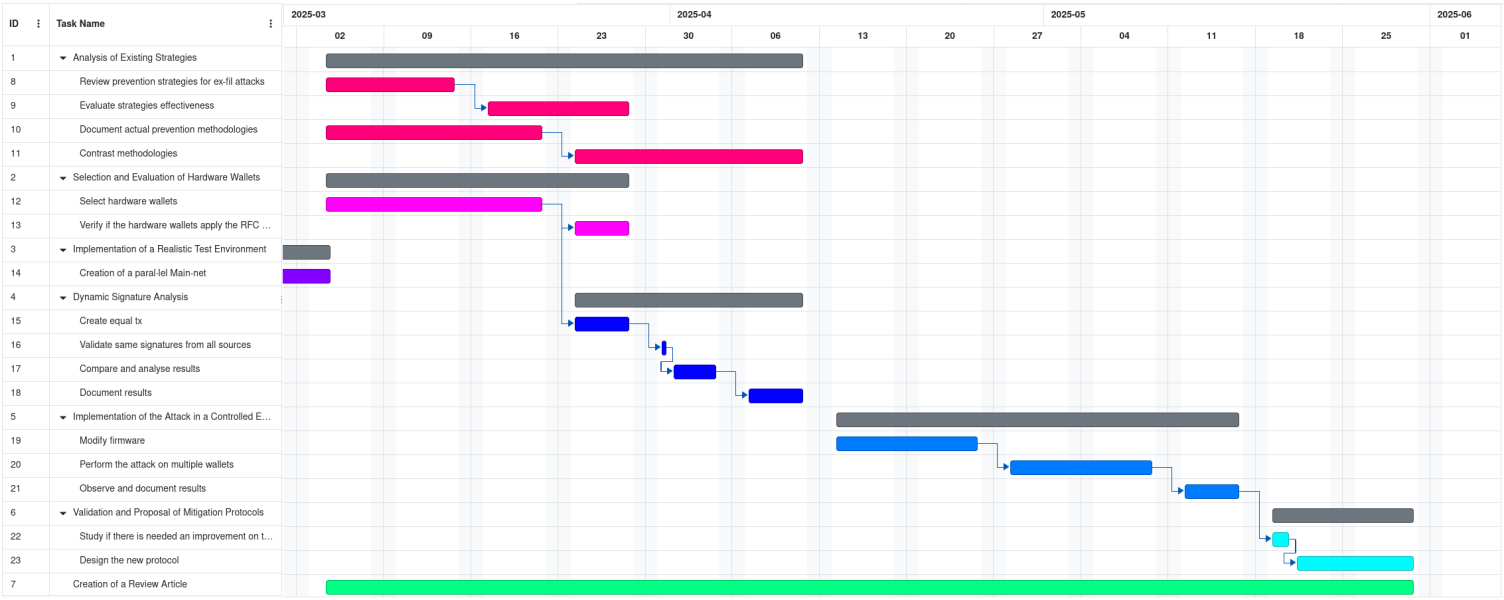


Figure 1: Gantt Chart

1.6 Ethical, Social, and Environmental Impact

With the adoption of Bitcoin as another type of asset with store-of-value properties, it is implicitly assumed that security and privacy matter to society. When talking about Bitcoin, a certain level of privacy and security is assumed due to its cryptographic complexity and the security measures embedded in its protocols. However, the highest level of security is ultimately at the user level. When various external companies offer devices related to Bitcoin but without inherently adopting the same high security standards, a problem arises. This creates a new attack vector that, external to Bitcoin itself, allows attackers to carry out malicious operations within the ecosystem.

The ethical responsibility within this project lies in following the principle of responsible disclosure, identifying potential issues related to covert channel attacks on these devices. Although many companies are already aware of these risks, providing synthesized knowledge or even proposing new solutions could be valuable.

Probably, the most crucial aspect to examine is the social dimension. Following Bitcoin's fundamental philosophy, as outlined by Satoshi Nakamoto in *Bitcoin: A Peer-to-Peer Electronic Cash System [16]:*

"What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing two willing parties to transact directly with each other without the need for a trusted third party. " Thus, ensure that each user is solely and entirely responsible for managing and securing their own assets. This project aims to strengthen this movement by providing reliability in these devices.

Regarding the environmental impact, while cryptocurrencies have been criticized for their energy footprint, this study is not directly related to the energy consumption of the network. However, implementing more efficient security protocols could help prevent the need to replace vulnerable devices, thus reducing electronic waste and the environmental impact associated with manufacturing new hardware.

2 Background

This work focuses on the analysis of covert channel attacks that exploit nonce values in probabilistic signature schemes implemented by hardware wallets used for Bitcoin management. Before delving into these specific attack vectors, it is necessary to establish a technical background on Bitcoin's authentication model. This includes a review of the mechanisms used to authorize the spending of funds, the cryptographic foundations behind them, and the vulnerabilities or limitations that may arise in practice.

2.1 Bitcoin digital signature schemes

In order to spend funds in Bitcoin, a transaction must be constructed that references one or more unspent transaction outputs, which, when referenced in another transaction, are included as inputs. Each of these inputs must be authorized by providing a valid digital signature that proves ownership to the spender that has the private keys used for "locking" the outputs.

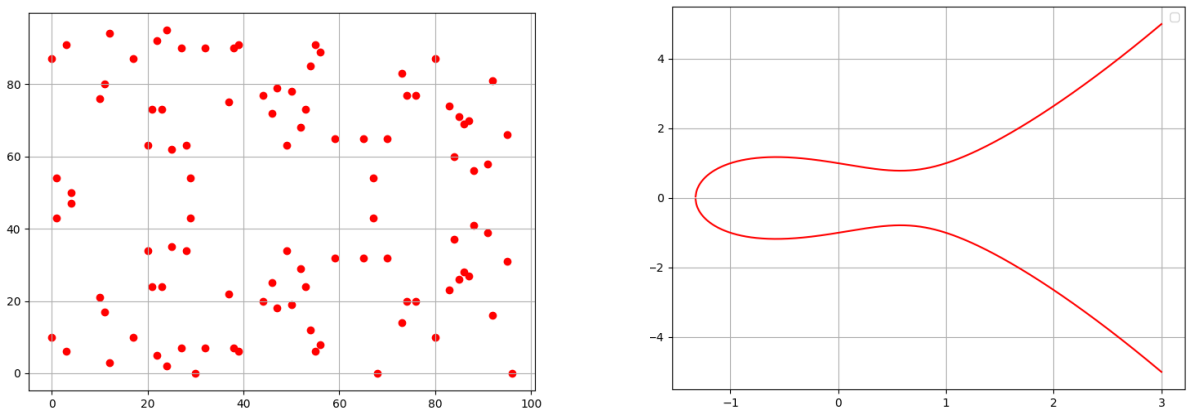
Since Bitcoin inception, two signing schemes have been introduced into the ecosystem. Although this study does not aim to explore these algorithms in depth, it is necessary to briefly outline some of their main characteristics to understand how Bitcoin applies digital signatures in its operations.

2.1.1 ECDSA

The standard signature scheme used in Bitcoin a few years ago was ECDSA [12], this scheme is based on elliptic curve cryptography (ECC) and is a variant of the digital signature algorithm (DSA), which has a similar process but does not work on a Galois field [25], also known as a finite field.

Elliptic curves are cubic equations in the plane that satisfy the following equation:

$$y^2 = x^3 + ax + b.$$



(a) Elliptic curve over F_{97}

(b) Elliptic curve over real numbers

Figure 2: Comparison of elliptic curves over finite field and real numbers

The points on this curve have coordinates defined over a finite field and must not intersect themselves or exhibit cusps. Cryptographic systems use finite fields because of their discrete nature and the computational hardness of problems such as the discrete logarithm. The graphical distinctions between an ECC over a finite field or real numbers are represented in Figure 2.

It is possible to add two points to the elliptic curve to obtain a third point within the same field, enabling the creation of secure values for digital signatures due to the elliptic curve discrete logarithm problem. This problem makes it difficult to retrieve the original scalar multiplied by the curve generator, which is used to compute the values r and s in the digital signature.

ECC offers some interesting properties because, although the security is equivalent to other schemes, the number of bits it works with is lower than, for instance, RSA.

ECDSA is probabilistic, which means that for the same message and private key, you may have different signatures due to an *ephemeral key* k which should be chosen randomly. If this k is not chosen randomly, some attacks can be made on this scheme in the event k is reused, as is described in Section 2.2.

To construct a signature with ECDSA it is mandatory first to define the domain parameters, which include the curve E/Z_p with module p and coefficient a and b, a generator point G that generates a cyclic group of order n. With those values, ECDSA pair key can be generated.

Pair key generation

1. Choose a random integer $d \in_R (1, n-1)$
2. Compute $P = dG$

Having d as the private key of the user and P as the point that works as the public key.

Signing The signature is generated with the user's private key.

1. Select a random integer $k \in_R (1, n-1)$.
2. Compute $R = kG = (x_1, y_2)$.
3. Compute $r = x_1 \mod n$. If $r = 0$, go to Step 1.
4. Compute $H(m)$.
5. Compute $s = k^{-1}(e + dr) \mod n$. If $s = 0$, go to Step 1.
6. The signature of the message m is (r, s)

Verification To verify a signature (r, s) for a message m, the verifier must have the domain parameters used to sign it and the public key of the signer.

1. Verify that r and s are integers between $[1, n-1]$.
2. Compute $e = H(m)$
3. Compute $w = s^{-1} \mod n$
4. Compute $u_1 = ew \mod n$ and $u_2 = rw \mod n$
5. Compute $R = u_1G + u_2Q$.
6. If $R = O$, then reject the signature. Otherwise, compute $v = int(x_1) \mod n$.
7. Accept the signature if and only if $v = r$.

2.1.2 Schnorr

Schnorr is an improvement over the ECDSA scheme. This digital signature was published by Claus Schnorr in 1990 in the paper Efficient identification and signatures for smart cards [21]. This scheme was patented over 20 years and therefore restricted to free use. The main benefit of the Schnorr scheme over ECDSA is its mathematical simplicity. Schnorr is based directly on the discrete logarithm problem on elliptic curves, something that ECDSA introduces with a more complex and not completely linear structure. In addition, the fact that Schnorr allows signature aggregation is a key point in its implementation in Bitcoin as it allows combining several signatures into one, also allowing to verify them in parallel in a more efficient way.

Pair key generation The key pair is generated in the same manner as described in Section 2.1.1.

1. Choose a random integer $d \in_R (1, n-1)$
2. Compute $P = dG$

Having d as the private key of the user and P as the point that works as the public key.

Signing As before, the process[1] involves the private key d and the message m.

1. In case y coordinate of P is odd $\Rightarrow d' = n - d$
2. Select a random integer $k \in_R (1, n-1)$.
3. Compute $R = kG = (x_1, y_2)$.
4. In case y the coordinate of R is odd $\Rightarrow k' = n - k$. Then, return to Point 3.
5. Compute the challenge $e = int(h(R_x||P_x||m)) \mod n$
6. Compute $s = (k + ed) \mod n$
7. The signature is (R, s)

Verification As in ECDSA, the verifier needs some specific public values to perform the validation.

1. Obtain the public key of the signer $P = dG$, the message m, and the signature (R, s).
2. Compute the challenge $e = int(h(R_x||P_x||m)) \mod n$.
3. Compute $R' = sG - eP$.
4. If the y coordinate of R' is odd, $\Rightarrow$ reject the signature.
5. Accept the signature if and only if $R'_x = R_x$.

2.1.3 Bitcoin Digital signature encoding

In the Bitcoin protocol, before a user can spend funds, a set of conditions encoded in the scripts must be satisfied. There are two types of script: those that lock funds (locking scripts, also called *ScriptPubKey* and those that unlock them (unlocking script, or *ScriptSig*). The locking script, attached to each unspent transaction output specifies the cryptographic conditions that must be met to authorize spending those coins, By the other hand, the unlocking script supplies the data that fulfills the requirements imposed by the locking script, ensuring that transaction is accepted as valid by network nodes. These scripts are fundamental in determining whether the spender possesses the correct private key associated with the UTXO locking script.

Over the years, both the conditions within the scripts and their placement within transactions have evolved. Below is a brief explanation of these developments.

Legacy encoding: These were the first types of locking script used in Bitcoin. They include *P2PK, P2PKH, P2MS, and P2SH.* All of them rely on *OP_ CHECKSIG*, which ensures that the output can only be spent by providing a valid public key corresponding to the private key that signed the transaction. All signatures done with these scripts are generated with **ECDSA**.

Segwit v0 encoding: This new algorithm was introduced in the Segregated Witness update [9], which brought significant changes, including a new transaction structure, an effective increase in block size and a new address format. The update also introduced *P2WPKH and P2WSH* two additional locking scripts, which are functionally similar to the legacy *P2PKH and P2SH* scripts, but are now unlocked using the transaction witness data.

As for the signature algorithm, it remains essentially the same as in the legacy system, though the data being signed differs due to the structural changes introduced by SegWit. Furthermore, it continues to use **ECDSA** as its primary signature scheme.

Segwit v1 (taproot) encoding: There are no official notes explaining why ECDSA was originally chosen for Bitcoin, but a likely reason is that Schnorr was in patent at the time. It was not until 2015, when Schnorr had already expired 5 years earlier, that some researchers started looking into introducing it into Bitcoin [28]. In 2021, the first signature algorithm using **Schnorr** was included in a Bitcoin update called Taproot [19]. This update introduced a new P2TR locking script that uses exclusively the Schnorr signature scheme.

[1] Steps 1 and 4 of the protocol are not standard for Schnorr signatures but reduce the public key size while maintaining a sufficient large key space.

2.2 Nonce reuse vulnerability in digital signature algorithms

Although this article explores a Bitcoin-specific solution to covert-nonce channel attacks. There exist some vulnerabilities related with the knowledge of the random value made to sign with the previously explained schemes. Once someone is able to know this value, they can easily discover the private key d that signed the message. Furthermore, if someone can detect that two different messages were made with the same private key d and the same nonce k, the private key could also be derived.

This vulnerability is neither new nor unique to Bitcoin, and is a well-known issue in probabilistic digital signature algorithms, and any system employing such algorithms is susceptible to this type of attack. By simply reusing a single nonce k in two different signatures, generated using the same private key d, but different messages m, an attacker can efficiently recover the key with relatively few computations. [31].

A notable real-world example is the 2010 PlayStation 3 security breach, where a flawed implementation of ECDSA by Sony allowed the hacker group fail0verflow to recover the company's private signing key. This enabled them to recreate the private key and break the signatures on signed executables [1].

This attack is already known to have occurred multiple times in Bitcoin. Although it could be due to various unknown poor implementations of some Bitcoin wallets, one well-known case is the Java Cryptography Architecture on Android versions prior to 4.4. In that version, the engineNextBytes method within the SecureRandom function in Apache Harmony 6.0M3 used a determinable randomness in case no seed was provided to the function [8].

Several studies such as the one conducted by Joachim Breitner and Nadia Heninger in their paper Biased Nonce Sense: Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies [5], or the user [OPSXCQ] on the [STRM] blog [18], determine that there were outputs on the blockchain containing Bitcoins that had been signed using the same nonces. In fact, these UTXO no longer exist as they were probably stolen, as discussed in the Bitcointalk forum [6].

It is important to emphasize that detecting whether a value k within an ECDSA signature has been reused is as simple as checking if the r value of the signature is the same in two different digital signatures, since $R = kG$ and $r = R_x$.

$$R = kG \mod n, \qquad s = k^{-1}(H(m) + dr) \mod n$$

In case a user creates two signatures with the same r value.

$$s_1 = \frac{m_1 + dr}{k}, \quad s_2 = \frac{m_2 + dr}{k}$$

$$s_1 \cdot k = m_1 + dr, \quad s_2 \cdot k = m_2 + dr$$

Subtracting both equations cancels out the shared term dr, leading to an equation that allows solving for k using the known values.

$$(s_1 - s_2) \cdot k = m_1 - m_2$$

$$k = \frac{m_1 - m_2}{s_1 - s_2}$$

By substituting k into the original signature equation, d can be isolated and solved.

$$d = \frac{s_1 \cdot (\frac{m_1 - m_2}{s_1 - s_2}) - m_1}{r} \mod n$$

$$d = \frac{m_1 s_2 - m_2 s_1}{r \cdot (s_1 - s_2)} \mod n$$

2.3 RFC 6979: a solution for nonce reuse

The RFC 6979 proposed by T.Pornin addresses a critical vulnerability in traditional ECDSA signature generation: the dependence on perfect randomness for the value k.

Quote that this proposal does not solve the nonce reuse problem, but implements a way to difficult the re-utilization of random values by implementing a mechanism that follows a deterministic approach. This mechanism ensures that, in the case that two signatures share the same nonce k they **must** (or with a really high probability) have the same hashed message $H(m)$ and the same private key d, providing highly unlikely accidental reuse.

RFC 6979 is based on **HMAC-DRBG** [3], it is used as a deterministic random bit generator based on HMAC [14], working $H(m)$ and d as the entropy introduced in the function. The 3 phases of the RFC are based on the HMAC-DRBG phases: Initialization, Generation and, lastly, Update. Figure 3 illustrates the HMAC-DRBG update function.

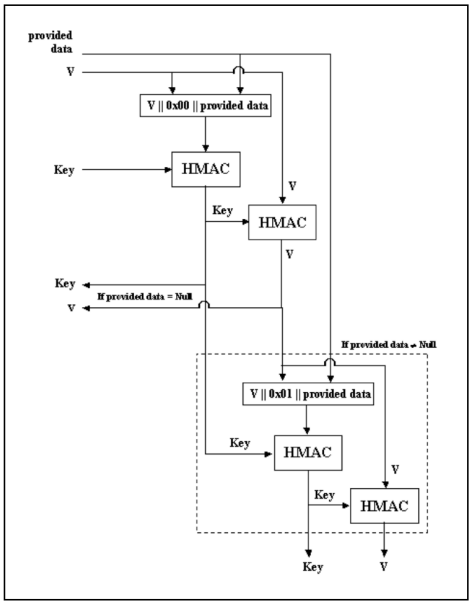


Figure 3: HMAC_DRBG Update Function, extracted from HMAC-DRBG paper [3]

.

Although similar, phases on deterministic nonce standard are not exactly the same. Each of them is explained in the next subsection.

2.3.1 Mathematical Generation Process

To better understand some of the later functions used in the creation of k, they are described below. For a more in-depth explanation, refer to the RFC 6979 standard [20].

$$\text{bits2int}(b) = \sum_{i=0}^{n-1} b_i \cdot 2^{n-1-i}$$

bits2int takes as input a sequence of blen bits and outputs a non-negative integer less than 2^{qlen}.

$$\text{int2octets}(x) = \text{big-endian representation of } x$$

In **int2octets** an integer value x less than n can be converted into a sequence of rlen bits, where $\text{rlen} = 8 \times \lceil \frac{\text{qlen}}{8} \rceil$, using big-endian encoding. The result is a sequence of octets.

$$\text{bits2octets}(b) = \text{int2octets}(\text{bits2int}(b) \bmod n)$$

The **bits2octets** transform takes a sequence of blen bits and outputs a sequence of rlen bits.

Initialization Phase Given:

- d: private key (integer $\in [1, n-1]$)
- m: given message (in Bitcoin, it is the pre-image of a transaction)
- $h = \text{bits2octets}(H(m))$
- q: group order

V and K values are settled, which will be the **initial vectors** of the function:

1. K is a a byte vector of `0x00` repeated *hlen* times.
2. V is a byte vector of `0x01` repeated *hlen* times.

Here, *hlen* is the length of the hash output. In Bitcoin 32 bytes, corresponding to the 32 bytes of SHA-256.

Generation Phase Once the internal state is initialized, the DRBG enters the generation phase. This phase is responsible for deriving a nonce k from the internal state vectors V and K. The process remains in a loop until a valid value k is found, k must belong to the cyclic group of order n and produce a value r different from zero, where r is derived from the x-coordinate of the point of the elliptic curve $R = kG \Rightarrow r = R[x] \mod n$.

Update Phase Finally, the update phase starts only when the candidate k produced during the generation phase is invalid. This phase updates the internal state of the vectors similar to the generation phase until the conditions are met.

2.3.2 RFC 6979 Pseudo-code for Deterministic Nonce Generation

The Algorithm 1 outlines the deterministic procedure defined by RFC 6979 to generate ephemeral values k using the HMAC mechanism. This approach guarantees that, for a given private key and message, the resulting k value is always the same, ensuring determinism in the signature process. Some implementations introduce additional steps during the initialization phase to incorporate more entropy into the internal state. In such cases, the determinism provided by the standard no longer holds, and the previously stated behavior would not apply, the resulting k value may vary across iterations, even when the same private key and message are used.

Algorithm 1 RFC 6979 - K generation process

Given the input message m, and $h = H(m)$
Where $r = R_x$ and $R = kG$
Phase 1: Initialization
$K = \text{HMAC}_K(V \parallel 0x00 \parallel \text{int2octets}(d) \parallel \text{bits2octets}(h))$
$V = \text{HMAC}_K(V)$
$K = \text{HMAC}_K(V \parallel 0x01 \parallel \text{int2octets}(d) \parallel \text{bits2octets}(h))$
$V = \text{HMAC}_K(V)$
Phase 2: Generation
while True **do**
  $V = \text{HMAC}_K(V)$
  $T = T \parallel V$ (append to T)
  $k = \text{bits2int}(T)$
  if $k \in [1, n-1]$ and $r \neq 0$ **then**
    generation is complete. Use the value k for the signature.
  else
    Phase 3: Update
    $K = \text{HMAC}_K(V \parallel 0x00)$
    $V = \text{HMAC}_K(V)$
  end if
end while

3 Covert-nonce channel attacks on Bitcoin

As described in the previous section, nonce reuse can be probabilistically mitigated by applying RFC 6979. Although this is a solid solution, it can still be bypassed if the attacker has access to the device where the signature is generated. By modifying the nonce, the attacker could embed a secret message that only those with privileged information could interpret. Since the value of k of the signature would be known, the attacker could then extract the private key from the device through a covert channel.

3.1 Bitcoin signing infrastructure: wallets

Bitcoins are managed using unspent transaction outputs (UTXO). Each Bitcoin transaction generates outputs that may later be used as input in new transactions. UTXOs are outputs that have not yet been spent and therefore remain available for use. In practice, what users actually control are these UTXOs, sending and receiving them as if they were coins to be spent.

Although in its early years, the Bitcoin infrastructure was operated primarily through command-line interfaces, today there are a variety of solutions that allow users to manage their UTXO in a more user-friendly and secure manner. These solutions are known as wallets and come in multiple types depending on the needs and cases of the users.

Hot wallets are typically software clients directly connected to the blockchain, allowing users to perform operations quickly and intuitively from their interface. They can also run in watch-only mode, which lets you observe all transactions associated with your addresses without ever storing the private keys locally. This is achieved by the BIP-32 deterministic wallet scheme proposed by Pieter Wuille [27], which supports hierarchical key derivation: the hot wallet holds only the public keys, while the corresponding private keys remain secure on an external device. Although keeping private keys off-wallet maximizes security, signing a transaction in watch-only mode becomes more cumbersome, since it must be done via an external signing tool. In contrast, classic hot wallets store private keys locally, allowing you to sign and broadcast transactions instantly, but at the expense of a reduced security boundary.

Cold wallets, also known as hardware wallets (HWW), are devices designed to securely store and manage user private keys in an offline environment, eliminating exposure to online threats. Unlike hot wallets, they do not have an internet connection, which means they cannot autonomously retrieve blockchain data or, even more importantly, broadcast transactions without physical access [2]. Instead, they rely on an external client software, which keeps track of account activity and facilitates user interaction. Having to manually verify transactions through the device before creating the signature ensures that only authorized operations are executed.

3.2 Adversarial model

It is crucial that hardware wallets are properly secured; otherwise, the private keys that custody the UTXOs would be exposed. If an attacker were to discover its secret value, they would be capable of executing transactions remotely, since they could authenticate as the rightful owners of the unspent outputs associated with those addresses, and further sending them to another private key they control.

Hot wallets are inherently more vulnerable as they are exposed to the Internet, the risk that the computer running the wallet software becomes infected with malware is a plausible scenario, and it is extremely difficult to mitigate all potential attack vectors. In this way, a user might believe that storing keys within a hardware wallet is the same as placing them inside an impenetrable fortress. However, it could not be further from the truth. There are certain attack vectors which, although their probability of occurrence is vanishingly small and require such dedication and expertise on the part of the attacker, being the target of one of them could be totally catastrophic, potentially resulting in the user losing all their coins in the last instance. Furthermore, these attacks are extremely difficult to detect, often only identifiable once the breach has already taken place.

What follows is an illustration of two such attacks that exemplify this category of highly potentially dangerous attack vectors:

- **Evil Maid Attack:** A type of attack in which the attacker physically gains access to a device while the legitimate user does not notice it. Gaining physical access to the device is an easy way to tamper with it, e.g. by installing malware, allowing the attacker to perform several malicious activities within the infected device.

- **Supply Chain Attack:** This attack takes place when the adversary compromises a device or software at some point within the supply chain or distribution process to insert malicious code before the product reaches the end user.

Attacks such as the 'evil maid' attack or 'supply chain' attack can be critical to the security of a hardware wallet. By tampering with the wallet's firmware, they can end up modifying its functionality in ways that, without privileged insider knowledge, are exceedingly unlikely to notice they are happening.

Notice that both attacks become feasible once an adversary briefly gains physical access to the hardware wallet. Although you cannot alter the onboard firmware after it has been installed and cryptographically verified on the device, many wallets require you to download firmware updates from the manufacturer's website. That download step opens up other attack vectors. For example:

- **Malicious firmware host:** If a user unknowingly downloads an update from a compromised or spoofed site, they can install malware in place of the legitimate firmware.
- **Man-in-the-middle (MITM):** An attacker that intercepts the user's connection to the firmware server could inject a modified firmware binary, which the wallet would then install unless the wallet enforces strong signature checks.

In both scenarios, the root of the problem is trusting an external source for firmware distribution without end-to-end verification. Ensuring that every update is signed by the manufacturer's private key and that the device strictly enforces signature validation eliminates these vectors.

3.3 Covert-nonce channel attacks

Some of the attacks mentioned in Section 3.2 can turn into sophisticated attacks in which the adversary compromises the device at the firmware level, allowing exploits to run without the user being aware of the danger. This work focuses on a specific attack that uses the random value used in probabilistic signatures to exfiltrate critical information from the HWW to an external source within a covert channel, such attacks are named covert-nonce channel attacks. The main idea behind them is to exchange information in an environment where communication is supposedly not possible, thereby violating the system's policies or security. In the Bitcoin environment, this attack can be carried out by transferring information from the cold wallet to an external environment due to the 'fully visible' nature of the blockchain through transaction signatures. This type of kleptographic attack was already anticipated as possible in 1997 when Adam Young et al. exposed in their paper *The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems* [34] a way to exchange information in a secret and invisible manner using the DSA algorithm, along with a few others.

In the context of this attack, there exist 3 key entities participating:

- **Hardware Wallets:** The device that holds the secret cryptographic material.
- **Software Wallets:** The interface or application responsible for communicating with the HWW and interacting with the blockchain network.
- **Adversarial Observer:** An external malicious party attempting to infer or extract information from the device.

Given these conditions, not only is the external observer not assisting with the protocol, but there could also be another malicious entity involved, for instance, a hardware wallet, software wallet, or both. In the latter case, protecting the user's secrets becomes unfeasible, as all entities are controlled by the same party.

Sections 3.3.1 and 3.3.2 describe two ways in which HWW are vulnerable to covert-nonce channel attacks. Although both extract sensitive information from the HWW, there are few differences between which information they are able to extract. While the next section discusses how a single private key could be extracted from the device with only one transaction, Dark Skippy explains that it is feasible to extract a Bitcoin seed of m bytes (which derives every other private key used for signing transactions) with only n signatures by solving a discrete logarithmic problem of $2^{\frac{8 \cdot m}{n}}$ bits of search.

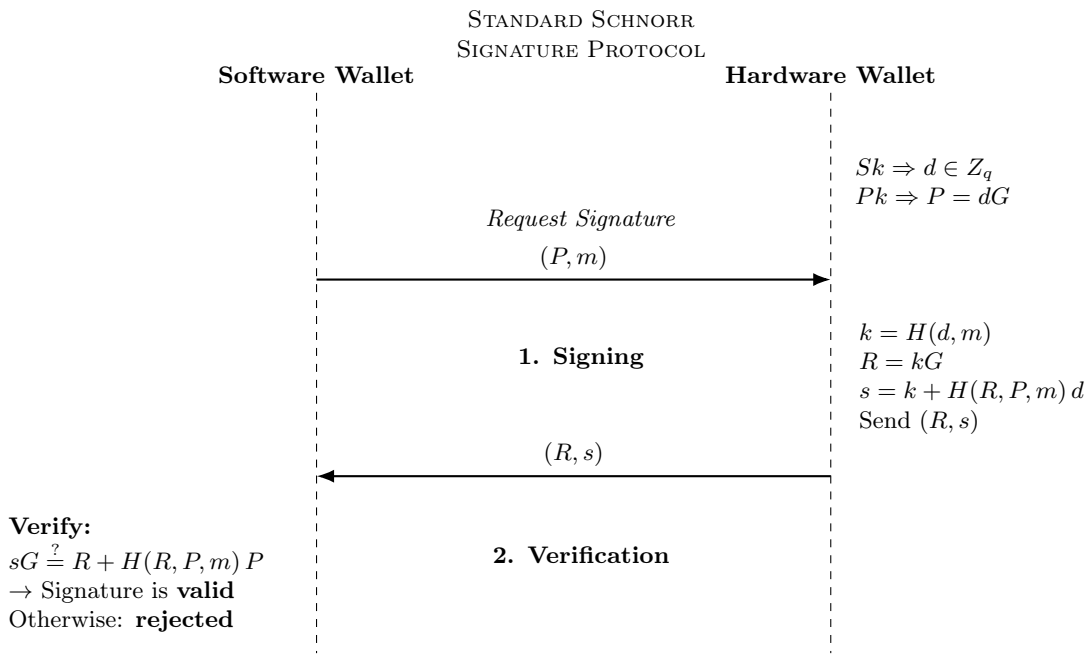


Figure 4: Standard Schnorr transaction signature protocol

3.3.1 Single private key filtering

As discussed in Section 2.2, knowing the value k of an ECDSA or Schnorr signature, someone could derive the private key used for its creation in a classical standard Schnorr transaction signature protocol (Figure 4).

This attack aims to have a covert-nonce channel between an external adversary and the HWW. To perform this attack, both should be the only ones who know a shared secret value z. It is important to note that although the value k may appear to be a random element, it is actually derived from the output of a hash function applied to the pair (m, d) following the RFC 6979 standard. However, this method of computation cannot be verified by a watch-only wallet, as it lacks access to the private key d.

This inability to verify the accuracy of the $k = H(d, m)$ computation makes the computation of $k = H(z, m)$ feasible. Since the adversary also knows z, they can independently compute the same k. This shared knowledge of k enables the external observer to recover the private key d using the relation:

$$d = \frac{s - k}{H(R, P, m)}$$

3.3.2 Dark Skippy

In this attack, Lloyd Fournier, Nick Farrow, and Robin Linus recently discovered that the seed extraction process through this attack can be completed in only a relatively few steps [15].

As stated on the disclosure website, '*Dark Skippy is a powerful method for a malicious signing device to leak secret keys.*' While this attack was disclosed on 8 March 2024, its potential risk had been known for years, as mentioned in a post on the Bitcoin Talk forum in 2014 [24].

The idea behind this attack is to tamper with the HWW firmware so that it becomes possible to track the transactions it creates on the blockchain, and then carry out a brute-force attack, assuming certain values used in the signature process are already known. This attack targets Bitcoin users who sign transactions using an infected cold wallet. Instead of generating a truly random nonce, the compromised device calculates it using part of the secret seed it is supposed to keep safe, combined with some secret controlled by the adversary. This process results in the leakage of a m byte seed just in n signatures. In the following example, it is assumed that the seed stored on the device is 16 bytes long, and the device leaks 8 bytes with each signature. On the other side, Figure 5 represents a 8 byte seed extraction with only 1 signature.

Attack Setup

1. The attacker compromises the signing device.
2. The compromised device generates random points $\mathbf{R}_i$, which are R_x the value r of an ECDSA signature with two key properties:
   - A verifiable **watermark** δ, used to iterate the generation of R until the desired value is found.
   - A **blind** value using a shared secret blinding factor $b_i = H(SECRET \mid\mid m \mid\mid i)$.

   $$k_0 = (seed[0:7] \cdot b_0)$$
   $$k_1 = (seed[8:15] \mid\mid \delta) \cdot b_1$$

   where m and $SECRET$ are the transaction ID and a secret chosen string, respectively. δ is an iterating value described in the following.
3. To embed the watermark, the malicious device performs a **grinding loop**, incrementing δ until $R_1 = k_1 G$ where $H(\text{SECRET} \mid\mid R_{1x})$ begins with `000...`. In this way, only the attacker can detect a corrupted transaction.
4. The transaction is broadcasted.

Attacker Detection

4. The attacker monitors network traffic and inspects transaction signatures.
5. For each signature, the public point $\mathbf{R_{1x}}$ is extracted and hashed to prove:

   $$h = H(SECRET \mid\mid R_{1x}) \stackrel{?}{=} \texttt{000xx...}$$

6. If the hash h matches the expected pattern, the attacker suspects a compromised transaction.

Nonce Unblinding

7. The attacker, knowing the blinding factor b, can unblind the nonce:

   $$b_0 = H(\text{SECRET} \mid\mid m \mid\mid 0)$$
   $$b_1 = H(\text{SECRET} \mid\mid m \mid\mid 1)$$
   $$R_i = b_i \cdot seed \ \cdot \ GD_i \Rightarrow D_i = b_i^{-1} R_i$$

8. After recovering the original nonces, the attacker is left with two discrete logarithm problems (one for each signature):

   $$D_0 = seed[0:7] \cdot G, \qquad D_1 = (seed[8:15] \mid\mid watermark - counter) \cdot G$$

9. Since only 64 bits of secret data are embedded in each nonce, the discrete log can be solved using algorithms such as **Pollard's Kangaroo [26]**:

$$\text{Expected complexity: } \sqrt{2^{64}} \approx 2^{32}$$

Seed Extraction

10. The attacker removes the eleven bits watermark counter from the nonce.
11. Both signature bytes are concatenated.
12. These bytes are then converted into a BIP39 seed phrase.
13. The attacker imports this phrase into a wallet and gains full access to the victim's funds.

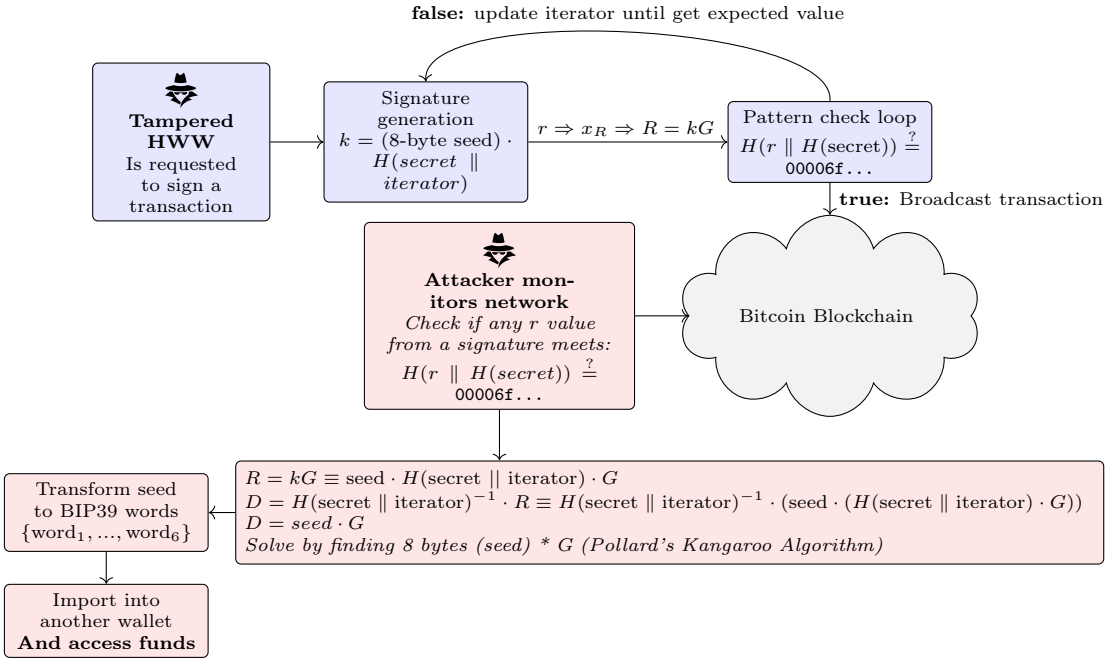


Figure 5: Dark-Skippy attack workflow with 8-byte seed

3.4 Solutions to covert-nonce channel attacks

The problem with these types of attack is that even if your system is completely air-gapped, you are still not protected, since it is a firmware-level issue related to how the signatures are generated.

To eliminate these risks, several mitigation techniques have been developed. Pieter Wuille illustrates this in a public email to the Bitcoin development mailing list [30], where he explains sequentially all the proposed schemes.

The intended solution for this type of attack is not to trust only your own device. Since these attacks are particularly difficult to detect, you need an external source of trust to ensure that your signature is legit.

3.4.1 Signature comparison between wallets

The first obvious solution is to validate a signature made with wallet A using another wallet B that employs deterministic signature generation. However, as will be explained later in Section 4.4.4, the wallet B must implement **low r grinding** only if the wallet A does, to ensure that the resulting signature is truly identical.

3.4.2 Single software wallet binding commitment

One of the solutions that implements this external source of entropy is a hot wallet, as depicted in Figure 6, where it provides additional entropy to the hardware wallet. This entropy should be incorporated into the nonce computation process.

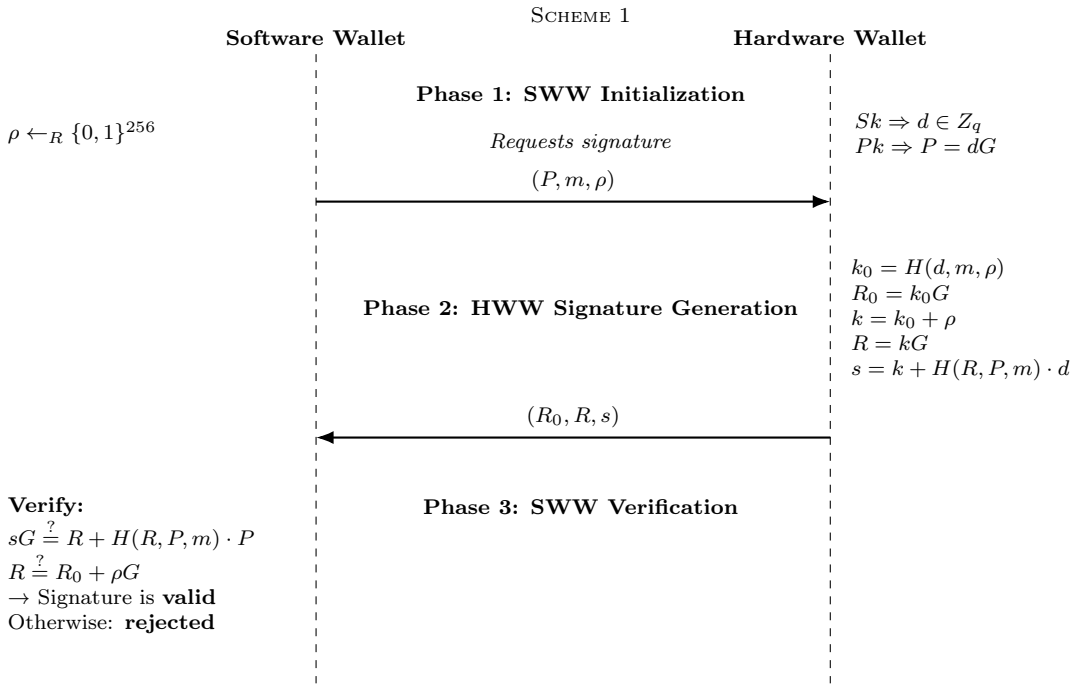


Figure 6: Signature generation using host-supplied entropy ρ

This solution does not fully address the problem, since the hardware wallet could still select a specific nonce value that suits its purposes and subsequently compute whether the validation on the software side $R = R_0 + \rho G$ satisfies the required conditions.

3.4.3 Deterministic nonce with sign-to-contract tweak

To address this issue, two potential solutions can be considered. The first solution is shown in Figure 7 and involves using sign-to-contract tweaking, a cryptographic technique that allows the commitment of a signature to an auxiliary value, in this case the ρ value provided by the software wallet. This approach was originally proposed by Greg Maxwell and discussed in the Bitcoin Forum [24]. Although at first seems a great solution. This scheme remains vulnerable to a different class of covert-nonce channel attacks. In this scenario, a malicious hardware wallet may attempt to embed bits of the master key into the signature by iteratively computing candidate values for k_0 and selecting those that cause m bits of the resulting signature to match m bits of the master key. With enough signatures, this covert channel could be exploited to leak the entire secret value.

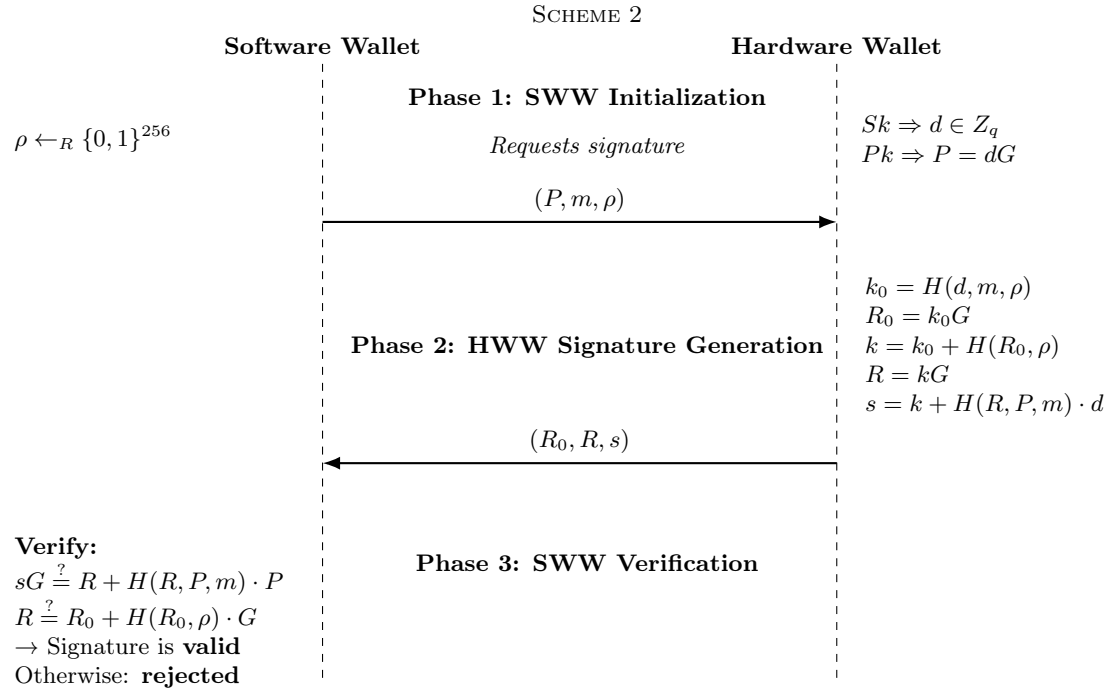


Figure 7: Signature generation with sign-to-contract tweak ρ

3.4.4 Deterministic nonce with tweak revealed after commitment

While Scheme 2 does not truly mitigate covert-nonce channel attacks, an alternative solution would be to force the hardware wallet to commit to a fixed nonce value $R_0 = k_0 G$. By doing so, the software wallet can verify that the nonce has not been subsequently manipulated, thereby ensuring the integrity of the signing process. Although the approach portrayed in Figure 3.4.4 appears to ensure that HWW cannot perform malicious actions, it still leaves room for SWW to exploit. Specifically, the SWW can manipulate the process by requesting two signatures on the same message m, while selecting two different values for ρ. Assuming that the HWW uses deterministic nonce generation, k_0 remains constant between the two requests.

$$R = (k_0 + t)G, \quad R' = (k_0 + t')G$$
$$s = k_0 + t + H(R, P, m) \cdot d, \quad s' = k_0 + t' + H(R', P, m) \cdot d$$

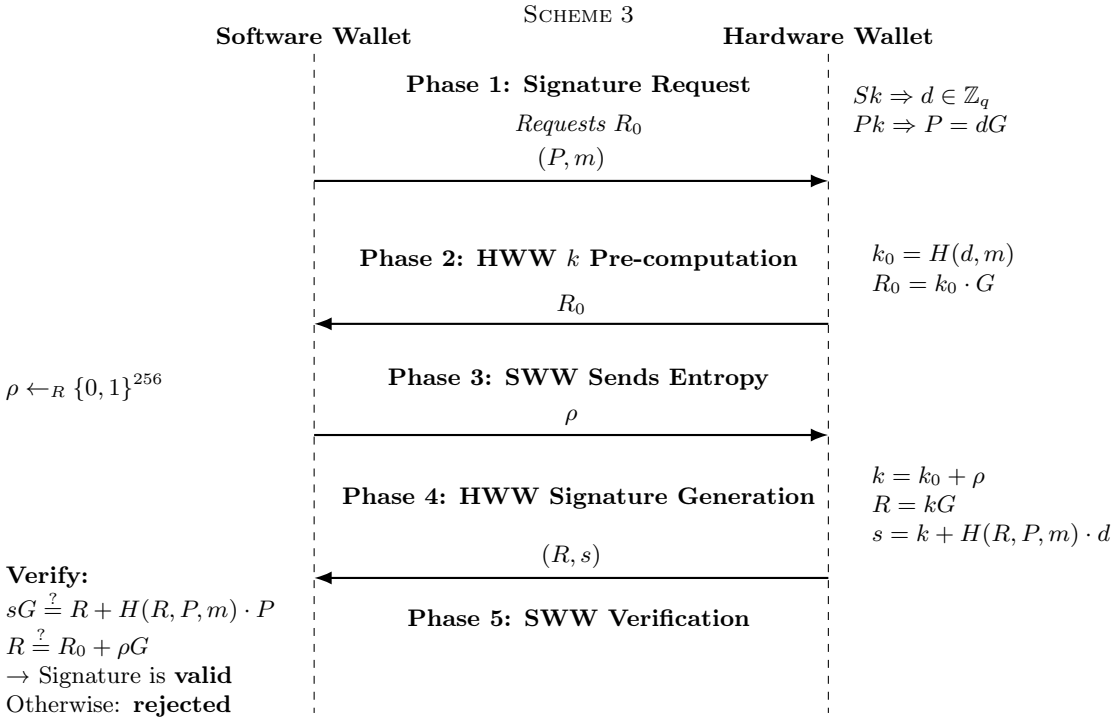
Since both signatures are computed over the same message m, the SWW can subtract the two equations s and s':

$$s' - s = (t' - t) + [H(R', P, m) - H(R, P, m)] \cdot d$$

Which rearranging the equation, d can be derived as:

$$d = \frac{s' - s - (t' - t)}{H(R', P, m) - H(R, P, m)}$$

This attack is functionally equivalent to a nonce reuse vulnerability, and equally, allows software wallet to steal all funds signed with private key d.



3.4.5 Signature generation using HWW randomness with tweak revealed after commitment

In Figure 8 a solution by Sergio Demian Lerner [24] is depicted. Lerner's proposal involves modifying the nonce k by introducing additional entropy, thereby ensuring that the resulting signature remains unique, even when the software wallet provides the same ρ value. Pieter Wuille later introduced an alternative scheme in the aforementioned Bitcoin mailing list, which modifies the protocol by changing the tweak applied to k from multiplicative to additive. It is important to note, however, that both of these solutions introduce non-determinism into the signing process, as they incorporate a varying component into the generation of the nonce.

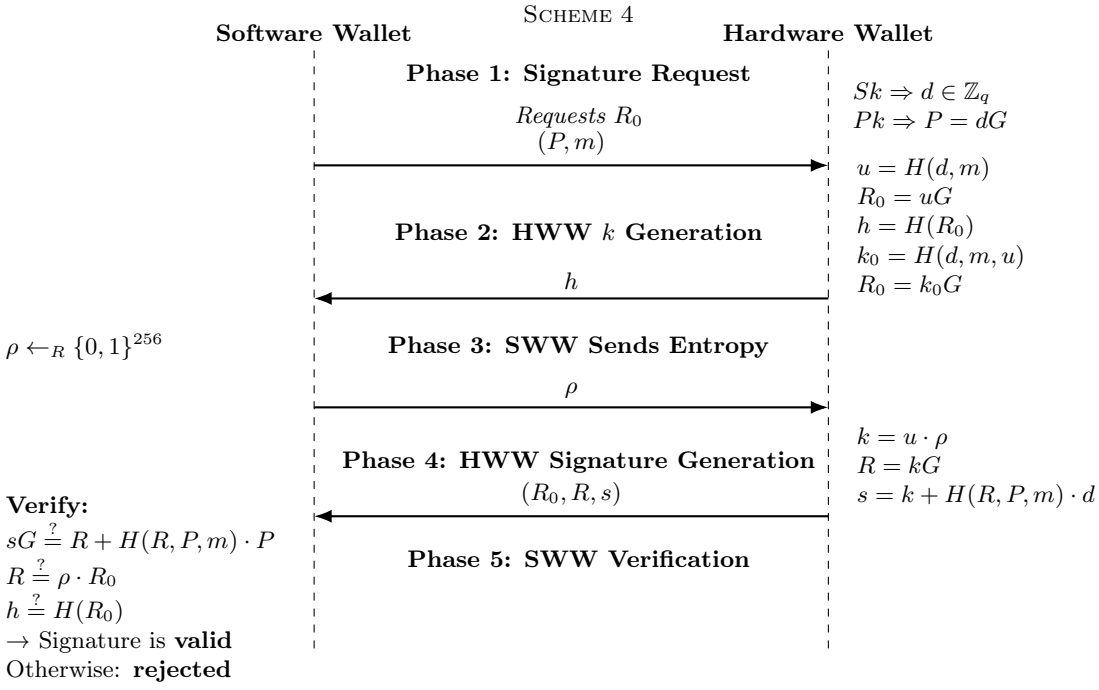


Figure 8: Signature generation using HWW randomness with tweak revealed after R_0 commitment

3.4.6 Deterministic signature generation with precommited tweak revealed after a blind commitment

Another way to ensure that the SWW does not alter the tweak after the protocol begins is to require it to commit to the tweak before revealing it. In Figure 9, originally suggested by Stepan Snigirev, this pre-committed tweak approach. In this scheme, the SWW first sends a hash commitment of the tweak to the HWW, which then responds with a commitment to its own derived nonce. Only after this exchange does the SWW reveal the actual tweak value, ensuring that both parties are bound to their respective commitments before any sensitive values are disclosed.

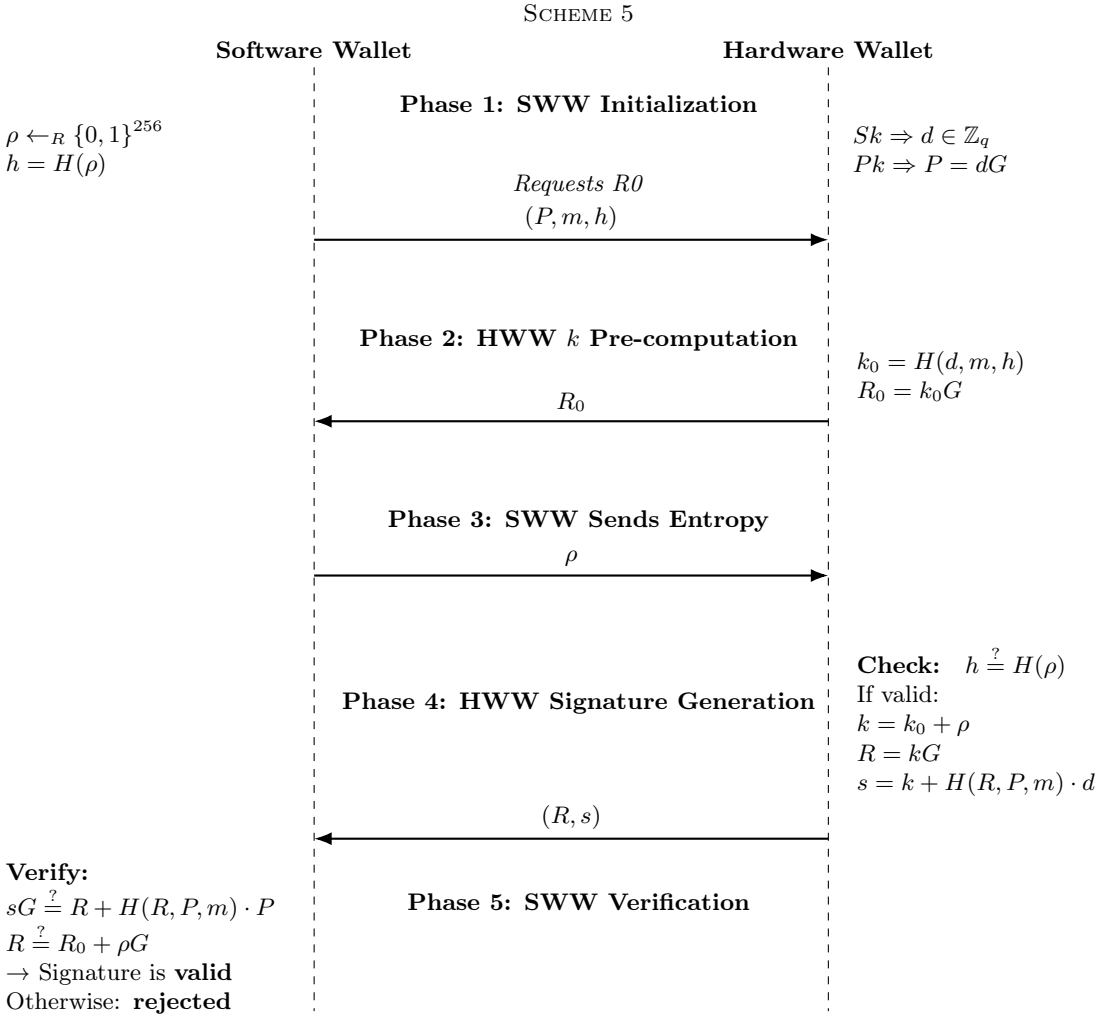


Figure 9: Deterministic signature generation with precommited tweak revealed after a blind commitment

Although Scheme 4 and Scheme 5 effectively mitigate covert-nonce channel attacks, their deployment on devices that interact with Bitcoin's off-chain infrastructure remains not trivial. This is mainly because most hot wallet implementations lack the verifier role, i.e. they are not generating and validating the cryptographic proofs that guarantee nonces have not been manipulated, and only a handful of vendor-specific software wallets (those paired with a proprietary cold-wallet app) provide this capability. As a result, this functionality is generally absent from the vast majority of software wallets. Furthermore, these schemes are not stateless, as they have to keep the value k_0 in memory between rounds, involving more problems to solve in implementation.

3.5 Anti-Exfil

The last proposed scheme has been Anti-klepto or anti-exfil [4], which is a protocol originally proposed by Andrew Poelstra and Jonas Nick. This protocol, as the other ones, aims to prevent the nonce used to generate a probabilistic digital signature from being manipulated to carry out attacks such as Dark Skippy.

The solution proposed by the developers of BitBox02 and Jade is to rely on your hot wallet, that is, the software wallet installed on your device e.g. laptop, smartphone, etc. to contribute a random value that will be combined with the nonce selected by the HWW. By mixing both nonces and later verifying that the provided random value was actually used, a user can be confident that, even if someone tries to manipulate it, the value is safe from modifications and cannot be known by the hot wallet.

The current implementation of this protocol follows the approach described in [33]. It should be noted that the initial implementation used was ECDSA [32] being slightly different since the generation of the signature is different itself. However, the process of calculating the tweaked nonce is performed in the same manner.

As depicted in Figure 10, the protocol begins with the SWW blindly adhering to a random value ρ using a hash function. Subsequently, the HWW responds with its own blind commitment to R_0. The final nonce used for the signature is then derived from both values, k_0 and ρ, ensuring that the resulting nonce is truly unpredictable and free from bias. This two-party commitment process strengthens the security of the signature by preventing either party from unilaterally influencing the nonce.

This makes it computationally infeasible for a compromised device to encode private key information in the signature without detection, as any manipulation would require cooperation between the host and the device or would be immediately detectable through signature verification failures.

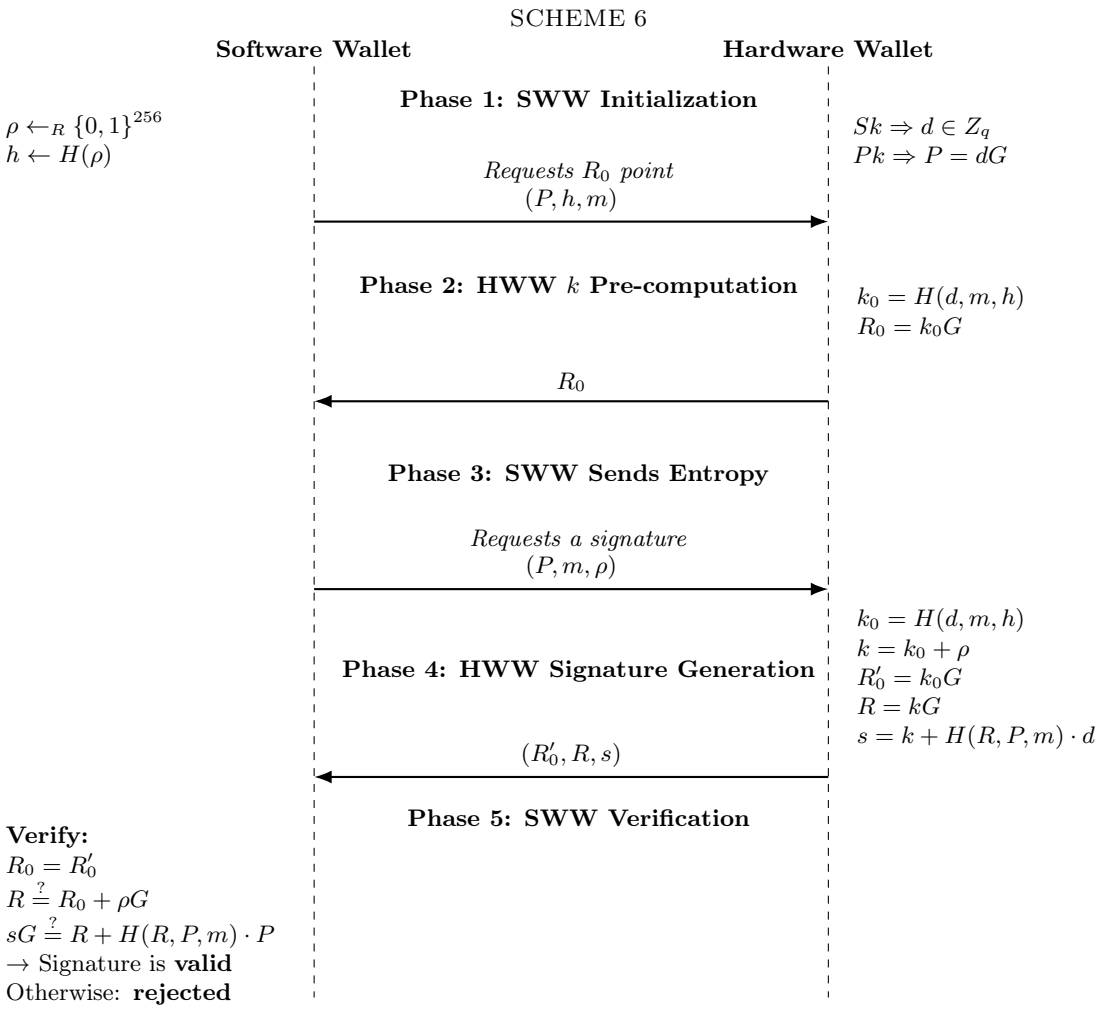


Figure 10: The anti-exfiltration proposal for Schnorr signatures

4 Experiments

The main objective of these experiments is to validate to what extent the proposed mitigations against nonce biasing or covert-nonce channel attacks are implemented. This involves exploring how hardware wallets generate ECDSA signatures and verifying whether the nonce generation adheres to the standards. Finally, investigate how feasible it would be for an attacker to carry out one of the attacks described in Section 3.4.

4.1 A mainnet test bed: Labnet

To enable unrestricted testing, a custom Bitcoin mainnet with the same properties as the original blockchain but without its historical data was created. This isolated network, built from a modified Bitcoin Core, allows for experimentation without spending real assets with hardware wallets that typically only support the real main-net.

To create this main-net, several essential components and configurations are required. The implementation has been carefully designed to ensure isolation from the real main-net.

4.1.1 Requirements

- **Bitcoin Core Source Code** The mainnet is built directly from Bitcoin Core's source code to allow modifications to key parameters, such as SegWit and Taproot activation heights or *minimumchainwork* option, among others. The chosen Bitcoin Core version was 24.0.1.
- **Docker** The entire network is deployed using Docker containers, which enable easy node creation, management, and isolation from external networks.
- **Python Mining Script or ASIC Miner** A simplified Python script is used to mine the initial blocks. For more efficient mining, an ASIC miner such as BitAxe (500 GH/s) can be used.
- **Internal Network Isolation** A dedicated Docker bridge network was created to ensure that the nodes remain isolated from the real Bitcoin network and external interference.

4.1.2 Implementation Strategy

- **Customizing Bitcoin Core:** Although it works exactly as the real one, the source code has to be modified, specifically the `chainparams.cpp` file to ensure that SegWit and Taproot are active from block 1. Bitcoin Core was compiled from the source with these modifications.
- **Setting Up Docker Network:** Created a dedicated bridge network using:

  ```
  docker network create -d bridge bitcoin_network
  ```

  Then, multiple Bitcoin Core nodes were deployed within containers.
- **Ensuring Synchronization:** At least two nodes were required to maintain proper synchronization. The router node handles internal communication between the network nodes.
- **Mining Blocks:** Initially, a Python script can be used to generate the first blocks, this Python script must generate the template as Bitcoin Core not return it with *getblocktemplate* RPC if the node is not completely synchronized. For long-term mining, an ASIC miner ensures efficient block generation.

4.2 Tested hardware wallets

Today, interested users have a wide variety of hardware wallets available on the market. Since it is unfeasible to perform tests with every single one to ensure the most realistic testing environment, the setup will involve acquiring some of the most widely used hardware wallets, simulating a real-world scenario, as these are the ones most commonly used by Bitcoin users.

The selection was based on research using Wallet Scrutiny [23], a resource that provides detailed evaluations of the available solutions on the market and highlights some of their most important technical issues. Preference was given to wallets that are Bitcoin-only and open-source (or at least open-to-read, if not fully open-source). The selected hardware wallets are the following:

- Ledger Nano S
- BitBox02
- Foundation Passport
- Trezor Safe 3
- Trezor Safe 5
- Coldcard Mk4
- Jade
- SeedSigner
- Keystone3 Pro
- BitKey
- SafePal S1
- KeepKey

To simplify the information in this report, Table 1 illustrates the most relevant features of each hardware wallet, along with a brief explanation of each.

Wallet	Bitcoin Only	Firmware Verification	Firmware Manual Install	Third-Party App Login	Volatile Memory	QR Code (Camera)	SD Card	Sparrow Integration	Open-To-Read
Ledger	✗	✓	✗	✗	✗	✗	✗	✓	✓
Bitbox02	✗	✓	✗	✗	✗	✗	✓	✓	✓
Foundation Passport	✓	✓	✓	✗	✗	✓	✗	✓	✓
Trezor Safe3	✗	✓	✗	✗	✗	✗	✗	✓	✓
Trezor Safe5	✗	✓	✗	✗	✗	✗	✗	✓	✓
Coldcard Mk4	✓	✓	✓	✗	✓	✓	✓	✓	✓
Jade	✓	✓	✓	✗	✓	✓	✓	✓	✓
SeedSigner	✓	✓	✓	✗	✓	✓	✓	✓	✓
Keystone 3 Pro	✗	✓	✗	✗	✓	✓	✓	✓	✓
Bit Key	✗	✓	✗	✗	✗	✗	✗	✗	✓
SafePal S1	✗	✓	✗	✓	✗	✗	✗	✗	✓

Table 1: Hardware wallet feature checklist

Feature	Description
Bitcoin Only	Supports Bitcoin exclusively (no altcoins).
Firmware Verification	Verifies that the installed firmware is authentic and unmodified.
Firmware Installation	Requires manual firmware installation or comes pre-installed.
Third-Party Login	Requires authentication via external accounts (e.g., APIs or trusted servers).
Volatile Memory	Does not store persistent data (all data wiped on reboot).
Air-Gapped	No direct connections (USB/Bluetooth); operates via QR codes or SD cards.
QR Code (Camera)	Scans QR codes using an integrated or external camera.
SD Card	Allows information transfer via SD cards
Sparrow Integration	Compatible with Sparrow Wallet (transaction import/export).
Open-To-Read	Code readable and auditable (typically with public GitHub repositories).

4.2.1 Wallet Connectivity Limitations:

Although most hardware wallets allowed the connection to standard hot wallets for main-net transactions, both SafePal S1 and BitKey presented unique restrictions due to their proprietary architectures. These devices required exclusive use of the hot wallet provided by the manufacturer, which enforced connections to dedicated node infrastructures. In both cases, the companion software implemented strict validation mechanisms that prevented wallets from connecting a custom private main-net.

4.3 Methodology

The tests performed to check signature generation on each HWW wallet followed the procedure below:

1. **Generated a single pre-image** (unsigned transaction):
2. **Imported each xpub from HWW to sparrow**.
3. **Signed it with multiple HWW** (all using the same private key).
4. **Analyzed the resulting signatures**, particularly the **DER-encoded** [10] signature format.

Example: SeedSigner output The signed transaction from SeedSigner was:

```
02000000000101389e3dafba72c444914ebfdf870f3726a955ca50674f6ebf6214d4324085615701
00000000fdffffff0200e1f505000000001600140e949fc32405cc43281ac2d0baef9e1b41321ac6
7383d71700000000160014bdbaeece25655c0734f46fdec28774476bec300e024730440220580a1d
4c313015ca7823ecfded5e7fdedfca0c0262d3b8eacba82b4e069a1d4002200f48c28ffa887a450d
78625bdf8a381bb65866464d818eaf0b36bdeded7c51f80121024ef0e172fb4f384073ca018d1985
d9a1515b2ab648fa0d339c62cbf332768efbe4000000
```

Here, the **DER-encoded signature** is:

```
30440220580a1d4c313015ca7823ecfded5e7fdedfca0c0262d3b8eacba82b4e069a1d4002200f48
c28ffa887a450d78625bdf8a381bb65866464d818eaf0b36bdeded7c51f801
```

4.4 Performed tests

When generating a Bitcoin payment transaction, a pre-image containing the transaction data (recipient, amounts, change address, fees, etc.) must be created before signing. This pre-image is converted into a Partially Signed Bitcoin Transaction (PSBT) and sent to hardware wallets for signing, either via physical connection or air-gapped methods.

Given that the signature creation process should be deterministic, following **RFC 6979**), the first hypothesis was that **all hardware wallets would produce the same signature** when signing the same PSBT with the same private key.

4.4.1 Initial Test

The objective was to observe whether the signature in the DER format, which, in the case of an **ECDSA** signature, consists of the values r and s, remains identical in all instances. To test this, a single pre-image was initially created, and later signed it with multiple hardware wallets with the same seed. This preliminary experiment was conducted to validate the hypothesis.

4.4.2 Result's discussion

Contrary to expectations, **three distinct signature groups** emerged:

1. **Group 1 (SeedSigner, Jade, Coldcard Mk4)**

   ```
   30440220580a1d4c313015ca7823ecfded5e7fdedfca0c0262d3b8eacba82b4e069a1d40022
   00f48c28ffa887a450d78625bdf8a381bb65866464d818eaf0b36bdeded7c51f801
   ```

2. **Group 2 (Keystone 3 Pro, Ledger Nano S Plus, Foundation Passport, Trezor Safe 3, Trezor Safe 5, KeepKey)**

   ```
   3045022100d9ba6c667de8ddf5b85fbe8717545da3b12a6e4851608788850281 23ec3518300
   2202a5daf4ddf18e00d2cec54aa50655b5cca34ac4a436e10949d533758419469d601
   ```

3. **Group 3 (BitBox02)**

   - Several signatures with this wallet were performed, in each of the results a **different signature was given**.

Initial observations from the results suggest that some hardware wallets **may not fully adhere to RFC 6979**, as their signatures did not consistently produce the expected deterministic value.

To rule out potential anomalies, the signing process using the same pre-image was repeated. Interestingly, Groups 1 and 2 produced identical signatures in both tests, **a sign of deterministic behavior, although possibly due to a custom implementation of RFC 6979**. In contrast, Group 3 generated distinct signatures each time, hinting at a non-deterministic approach.

To verify these findings, additional exhaustive tests were performed.

4.4.3 Exhaustive Tests

To verify whether modifications in input/output fields affected the deterministic behavior of the algorithm, five test scenarios with varying transaction structures were performed. Since RFC 6979 derives the nonce from the private key d and the message hash $h(m)$, these structural changes should not alter the resulting signature. The test cases were designed as follows:

- 1 Input – 1 Output
- 1 Input – 2 Outputs
- 3 Inputs – 2 Outputs
- 3 Inputs – 1 Output
- 1 Input – 3 Outputs

For each transaction, the DER encoded signature was extracted and later decoded into its constituent r and s ECDSA components. The results showed negligible variation across all configurations, which is a strong indicator that the wallets correctly isolate the message hash from the transaction formatting.

Given this consistency, the analysis is focused on the **1 Input − 1 Output** (see Table 2) case as a representative example, noting only minor deviations in other scenarios, which will be explained later.

Table 2: 1 Input - 1 Output **r** & **s** ECDSA signature hexadecimal values

Wallet	**r (bytes)**	**s (bytes)**	`r value`	`s value`
Ledger Nano S Plus	33	32	`00aba...d7b`	`1f4d0...df5`
Bitbox	33	32	`0084a...acd`	`3a7d6...1be`
Foundation Passport	33	32	`00aba...d7b`	`1f4d0...df5`
Trezor Safe 3	33	32	`00aba...d7b`	`1f4d0...df5`
Trezor Safe 5	33	32	`00aba...d7b`	`1f4d0...df5`
Coldcard Mk4	32	32	`635c4...8a9`	`5b029...90d`
Jade	32	32	`635c4...8a9`	`5b029...90d`
KeepKey	33	32	`00aba...d7b`	`1f4d0...df5`
Keystone 3 Pro	33	32	`00aba...d7b`	`1f4d0...df5`
Seed Signer	32	32	`635c4...8a9`	`5b029...90d`

4.4.4 Result's discussion

As expected, the initial test results aligned closely with the exhaustive tests. However, conducting these additional experiments provided valuable insight into signature behaviors.

A consistent pattern was observed where the value r occasionally required a leading zero byte in its DER encoding, which occurred precisely when r resided in the upper half of the range (most significant bit = 1). This encoding requirement stems from DER's signedness rules, where integers must be represented as unsigned values, necessitating the zero-byte prefix when the most significant bit would otherwise suggest a negative two-complement interpretation.

Although the value of s consistently remains in the lower half of the curve range [13] to prevent signature malleability attacks [29], this artificial constraint does not apply to r. The natural distribution of r values across both range halves consequently produces variable-length signatures, with approximately 50% requiring an additional byte.

Further investigation revealed that several hardware wallets, particularly those in Group 1, implement a distinctive approach to generate the r value. Although they adhere to RFC 6979 for nonce deterministic generation, they employ an additional optimization called *Low-R Grinding*. This technique involves iterating the signature process until a value of r is obtained in the lower half of the curve range, removing 1 byte from the final signature length.

Bitcoin uses DER encoding to express ECDSA signatures in Bitcoin. In that type of encoding, if the r or s values of the signature begin with a byte of 0x80 or higher, they must be preceded by a byte of 0x00. Therefore, some wallets implement strategies to repeat the signing process when the point related to the value of r falls within the upper range of the finite field, which occurs with a expected probability of 1/2. This simple technique avoids the extra byte in the final signature representation by simply repeating the operation until a lower value is found. In Table 4.4.4, two different signatures generated from the same transaction are shown. The variation in their r and s values is due to whether the signing wallets implemented low-r value grinding or not.

Label	Bytes	Hex Value
r	33	`00aba45accb4b9d6daa5e21d0cf384ac93a5c9c91b2874dcb7facd85e710db0d7b`
s	32	`1f4d0d1301ac321ba0ce62ca0afc0707b096afc8388a5e1865d649d00ddf6df5`
r	32	`635c416284dfededcd5a1aa1ad0b52c857a370a218b0b92c00940897fa7e038a9`
s	32	`5b02945e2b1a636c8cf575d9dd85398b1d22554c5746aacf07c5c92ccd46790d`

Table 3: Example of low r grinding from same transaction signature with different devices.

The inconsistent signature behavior observed in Group 3 (BitBox02) is due to its implementation of the **Anti-Exfil** protocol.

Although each transaction signature differs when signing with BitBox02, the process does not correspond exactly to the diagram in Section 3.5, since the modifications appear to occur only at the hardware wallet level. The observed procedure was as follows: to sign a transaction with BitBox02, one must first launch the BitBox02 App and physically connect the hardware wallet. At this stage, the app sends random data to the device, and the device uses those data, together with its own entropy, to generate a unique nonce k for the signature. In other words, the device is indeed implementing Anti-Exfil, but only on its own side. Sparrow Wallet, however, does not implement any Anti-Exfil protocol on the software side, so it cannot validate that the nonce k has been generated correctly.

Similarly, although the Jade feature page states that Jade also supports Anti-Exfil, this appears to require two conditions: (1) the device must be physically connected to the computer and (2) must use Blockstream Green wallet. Attempts to invoke this feature from Sparrow Wallet have failed. For example, in an experiment designed to create two 'identical' transactions and compare their signatures, Blockstream Green would not reveal the raw, signed transaction data before broadcasting. Once the first transaction was broadcast, its UTXOs became spent, making it impossible to create the same transaction the second time. Consequently, without support form software side for Anti-Exfil, the protocol cannot be fully trusted because there is no validation that the hardware wallet has combined its nonce correctly and has not leaked any secret information.

4.4.5 Nonce-related mechanisms in Hardware Wallets

Table 4 summarizes whether hardware wallets implement the RFC 6979 standard, anti-exfiltration protocols, or low-r grinding features discussed in this document.

Hardware Wallet	RFC6979 Implementation	Anti-Exfil	Low-R Grinding
Ledger Nano S Plus	Yes (custom RFC6979)	No	No
Bitbox	No	Yes	No
Foundation Passport	Yes (custom RFC6979)	No	No
Trezor Model T (Safe 3)	Yes (custom RFC6979)	No	No
Trezor Model T (Safe 5)	Yes (custom RFC6979)	No	No
Coldcard Mk4	Yes (secp256k1)	No	Yes
Jade	Yes (secp256k1-zkp)	Yes	Yes
KeepKey	Yes (Trezor library)	No	No
Keystone 3 Pro	Yes (secp256k1)	No	No
Seed Signer	Yes (Embit)	No	Yes

Table 4: Comparison of RFC6979, Anti-Exfil, and Low-R Grinding Implementations in HW Wallets

5 Conclusions and further research

The present study set out to explore state-of-the-art practices and technologies related to Bitcoin digital signatures, focusing specifically on nonce generation schemes and their vulnerabilities.

First, to facilitate controlled experimentation and ensure security testing in a realistic environment, a dedicated Bitcoin mainnet testbed was created. This environment simulates real-world blockchain behavior while allowing for complete control over network parameters and interactions. This setup enabled for testing without compromising actual assets, depending on external blockchain or spending any real capital.

In fact, that lab-net allowed us to carry out evaluations on whether the wallets tested follow RFC 6979, the standard deterministic nonce generation for digital signatures. It was confirmed that, while this standard is generally applied, some hardware wallets use custom implementations. Despite differences in code, these implementations still produce the same deterministic result, which is critical for maintaining nonce reuse protection. Some of them showed slightly different results due to the implementation of an optimization technique that forces the deterministic algorithm to perform more rounds than usual, which could end up having a different signature. These implementations iterated until they found a k such that its corresponding point R on the elliptic curve lies in the lower half of the curve. This optimization is designed to produce 1-byte less signatures.

In addition, a detailed review was also conducted on internal firmware implementations of various hardware wallets, particularly focusing on how they implement countermeasures against covert-nonce channel attacks. The study explored the different ways an attacker could potentially exfiltrate sensitive information through covert channels using hardware wallets. It was observed that in cases where such channels exist, they are extremely difficult to detect without shared privileged information between the adversary and the hardware wallet. It was found that, while some hardware wallets do implement mitigation protocols, the protection is not fully done unless the corresponding software wallet from the same organization is used. This is because validation of nonce generation on these protocols, requires collaboration between both the hardware and software sides. Without that complementary software, the validation process remains incomplete.

Based on documented attacks in the literature, the resources and time needed to extract a private key was estimated using this method. Moreover, it was illustrated that such an attack could go beyond simply compromising a single key. It could potentially exfiltrate the wallet seed itself. Since the seed is used to derive all the keys within the wallet, its exposure would result in a total compromise of all associated UTXOs, effectively giving an adversary full access to the wallet's contents. Our analysis shows that, although still challenging, a covert-nonce attack is significantly more feasible than solving a discrete logarithmic problem 2^{128}. In fact, the key could be recovered by solving two subproblems each of complexity of the order of 2^{32}, drastically increasing the viability of the computation.

To ensure a user's device is not being exploited, some best practices have been highlighted, such as verifying downloaded firmware using GPG signatures, a process already followed by several manufacturers but still worth reinforcing as a security measure. When a hardware wallet implements anti-exfiltration, be sure to use its specific software wallet app. In addition, compare signatures between two hardware wallets from the same manufacturer. Use hardware wallets in an air-gapped setup, and, of course, avoid reusing Bitcoin private keys.

Lastly, the partial study conducted in this project reveals several critical issues and sets the foundations for future research aimed at enhancing the security of Bitcoin hardware wallets against covert-nonce channel attacks. Although HWWs are designed to be a cryptographic stronghold to all forms of attack since they are specially security designed and isolated from external networks, our results show that they remain vulnerable to certain sophisticated threat vectors.

For future directions, a key focus should be on ensuring the implementation of anti-exfil protocols on both the hardware and software sides, particularly for open-source solutions where such mechanisms are often absent. Another major challenge identified is the inefficiency of current anti-exfil protocols, which require multiple rounds of communication, an especially tedious requirement for air-gapped devices. Thus, there is a pressing need to develop new approaches that can achieve the same security guarantees with a reduced number of interaction rounds, making them more practical for use in the real world.

Acknowledgments

To Cristina Pérez, for her dedication to reviewing each version of this work and for her constructive advice that helped me improve at every stage.

To Jordi Herrera, who guided and supported me in this project, providing clarity and insight from the beginning.

To my colleagues from the Discrypt research group for their ideas, feedback and technical support during key moments.

And to my family and friends for dedicating their time and smiles to listening to me talk about "weird Bitcoin stuff."

References

[1] 1KIND Tech. Dcemu reviews - 27c3 - chaos communication congress 2010 - fail0verflow - full video, 2023. Accessed: 2025-05-26. URL: https://www.youtube.com/watch?v=4loZGYqaZ7I.

[2] Myrto Arapinis, Andriana Gkaniatsou, Dimitris Karakostas, and Aggelos Kiayias. A formal treatment of hardware wallets. Cryptology ePrint Archive, Paper 2019/034, 2019.

[3] Elaine Barker and John Kelsey. Recommendation for random number generation using deterministic random bit generators. Special Publication 800-90A Revision 1, National Institute of Standards and Technology (NIST), 2015.

[4] Blockstream. Anti-exfil: Stopping key exfiltration. Blockstream Engineering Blog, February 2021.

[5] Joachim Breitner and Nadia Heninger. Biased nonce sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies. Cryptology ePrint Archive, Paper 2019/023, 2019. URL: https://eprint.iacr.org/2019/023.

[6] BurtW. Bad signatures leading to 55.82152538 btc theft (so far). https://bitcointalk.org/index.php?topic=271486.0, 2013. Accessed: 2025-05-26.

[7] Bitcoin Stack Exchange Discussion. What exactly is the benefit of buying a hardware wallet for bitcoin?, 2020. Accessed: 2025-03-04. URL: https://bitcoin.stackexchange.com/questions/99646/what-exactly-is-the-benefit-of-buying-a-hardware-wallet-for-bitcoin.

[8] Felix Dörre and Vladimir Klebanov. Practical detection of entropy loss in pseudo-random number generators. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS)*, pages 1101–1112. ACM, 2016. URL: https://formal.kastel.kit.edu/klebanov/pubs/ccs2016.pdf, doi:10.1145/2976749.2978369.

[9] Pieter Wuille Eric Lombrozo, Johnson Lau. Segregated witness, December 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki.

[10] ITU-T. X.690: Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), 2002.

[11] Briony J.Oates. *Researching Information Systems and Computing*. SAGE, 2006.

[12] Donald Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1:36–63, August 2001. doi:10.1007/s102070100002.

[13] Pieter Wuille Johnson Lau. Bip-146: Dealing with signature encoding malleability. Bitcoin Improvement Proposal, May 2016. Accessed: 2025-03-30. URL: https://github.com/bitcoin/bips/blob/master/bip-0146.mediawiki.

[14] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Hmac: Keyed-hashing for message authentication. Request for Comments 2104, Internet Engineering Task Force (IETF), February 1997.

[15] R. Linus L. Fournier, N.Farrow. Dark skippy disclosure, 2024. URL: https://darkskippy.com/.

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

[17] NBC News. Bitcoin could hit $100,000 by end of 2024, analysts predict, 2024. Accessed: 2025-03-04. URL: https://www.nbcnews.com/business/markets/bitcoin-100000-rcna181008.

[18] OPSXCQ. Bitcoin transaction nonce reuse vulnerability. https://strm.sh/studies/bitcoin-nonce-reuse-attack/, 2018. Accessed: 2025-05-26.

[19] Jonas Nick Pieter Wuille and Anthony Towns. Taproot: Segwit version 1 spending rules, January 2020. URL: https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki.

[20] T. Pornin. Deterministic usage of the digital signature algorithm (dsa) and elliptic curve digital signature algorithm (ecdsa). Request for Comments: 6979, August 2013.

[21] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1990. URL: https://link.springer.com/content/pdf/10.1007/0-387-34805-0_22.pdf, doi:10.1007/0-387-34805-0_22.

[22] Stepan. Hardware wallets can be hacked, but this is fine. CryptoAdvance on Medium, 2019. Accessed: 2025-03-04. URL: https://medium.com/cryptoadvance/hardware-wallets-can-be-hacked-but-this-is-fine-a6156bbd199.

[23] WalletScrutiny Team. Walletscrutiny - reviews of bitcoin wallets. Online, 2021. Accessed: 2025-03-30. URL: https://walletscrutiny.com/.

[24] Forum users. A covert-channel-free black-box signer without znps, December 2014. URL: https://bitcointalk.org/index.php?topic=893898.msg9841502#msg9841502.

[25] Wikipedia. Finite field. Accessed: 2025-03-31. URL: https://en.wikipedia.org/wiki/Finite_field.

[26] Wikipedia. Pollard's kangaroo algorithm. https://en.wikipedia.org/wiki/Pollard%27s_kangaroo_algorithm, 2025. Accessed: 2025-04-29.

[27] Pieter Wuille. BIP 0032: Hierarchical deterministic wallets. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki, 2012. Bitcoin Improvement Proposal.

[28] Pieter Wuille. Tree signatures: Multisig on steroids using tree signatures, August 2015. Accedido el 22 de mayo de 2025. URL: https://decred.org/research/wuille2015.pdf.

[29] Pieter Wuille. In ecdsa, why is r,-s mod n complementary to r,s? Bitcoin Stack Exchange, Mar 2019. Accessed: 2025-03-30. URL: https://bitcoin.stackexchange.com/questions/83408/in-ecdsa-why-is-r-%E2%88%92s-mod-n-complementary-to-r-s.

[30] Pieter Wuille. Overview of anti-covert-channel signing techniques. Bitcoin Mail List, 2020. Accessed: 2025-04-28.

[31] Pieter Wuille. How do you derive the private key from two signatures that share the same k value?, 2025. Accessed: 2025-03-30. URL: https://bitcoin.stackexchange.com/questions/73622/how-do-you-derive-the-private-key-from-two-signatures-that-share-the-same-k-valu.

[32] Pieter Wuille, Andrew Poelstra, and Jonas Nick. ecdsa sign-to-contract module, with anti nonce covert chan util functions, October 2019. URL: https://github.com/bitcoin-core/secp256k1/pull/637/commits/962ce23f8904f4ebd50859bdf1a1d3d49d84b497.

[33] Pieter Wuille, Andrew Poelstra, and Jonas Nick. Schnorr sign-to-contract and anti-exfil, October 2024. URL: https://github.com/bitcoin-core/secp256k1/pull/1140/commits/49c31379082563efdf858072f6241d4cfbbf44d6.

[34] Adam Young and Moti Yung. The prevalence of kleptographic attacks on discrete-log based cryptosystems. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 264–276, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[35] Tao Zhang, Bingyu Li, Yan Zhu, Tianxu Han, and Qianhong Wu. Covert channels in blockchain and blockchain based covert communication: Overview, state-of-the-art, and future directions. *Computer Communications*, 205:136–146, May 2023.