The curious case of the half-half Bitcoin ECDSA nonces

Dylan Rowe[1], Joachim Breitner[2][0000−0003−3753−6821], and Nadia Heninger[1][0000−0002−7904−7295]

[1] University of California, San Diego
drowe@ucsd.edu,nadiah@cs.ucsd.edu
[2] Unaffiliated
mail@joachim-breitner.de

Abstract. We report on a new class of ECDSA signature vulnerability observed in the wild on the Bitcoin blockchain that results from a signature nonce generated by concatenating half of the bits of the message hash together with half of the bits of the secret signing key. We give a lattice-based attack for efficiently recovering the secret key from a single signature of this form. We then search the entire Bitcoin blockchain for such signatures, and identify and track the activities of an apparently custom ECDSA/Bitcoin implementation that has been used to empty hundreds of compromised Bitcoin addresses for many years.

1 Introduction

It is well known in the cryptography community that the ECDSA signature scheme is *fragile* against nonce generation vulnerabilities. An attacker can recover a signer's ECDSA private key if they know the nonce used to generate a single signature; if a signer signs two distinct messages with the same nonce; if a signer signs multiple messages with unexpectedly short nonces; if the attacker can learn the most significant bits of many signature nonces, and so on.

In this paper, we report on an apparently new class of vulnerable ECDSA signature nonces that we discovered in the wild on the Bitcoin blockchain: nonces k of the form

$$k = h_{\text{msb}} \,||\, d_{\text{msb}}$$

where h_{msb} are the most significant half (128 bits) of the transaction hash, and d_{msb} are the most significant half (128 bits) of the signer's private key.

We give a lattice-based algorithm to recover the private key from a *single signature* generated with a nonce of this form in around one core-second of computation time, with almost 100% success rate. While our attack is a variant of existing ECDSA key recovery attacks, the particular vulnerability, the problem formulation we give to exploit it, and the fact that it is exploitable with only a single signature appear to be new observations.

We then search the Bitcoin blockchain for signatures of this form and find nearly 90,000 such signatures generated by around 900 addresses. These signatures have been in use from 2015 until now, and have been used to move 222 Bitcoin.

Nearly all of the transactions emptied funds from addresses whose private keys had been exposed on the web in some fashion: brainwallets with compromised passwords, pathologically short private keys, addresses given as examples in online documentation, and so on.

We hypothesize that these signatures are an artifact of a custom ECDSA implementation used by a thief to steal these funds. A number of Bitcoin forum postings link addresses receiving these funds to a particular individual who is relatively public about these activities, in addition to other scams.

While significant attention has been paid in the literature to using transaction graph analysis to trace funds and deanonymize users on the Bitcoin network, our work illustrates a case of a novel cryptanalytic attack allowing us to identify a peculiar ECDSA implementation mistake that apparently allows unique identification of a malicious user.

2 Background and Related Work

2.1 Bitcoin

Bitcoin uses the Elliptic Curve Digital Signature Algorithm (ECDSA) to authenticate the sending party of a transaction. These transactions are validated by nodes in the network and published publicly on the Bitcoin blockchain. A Bitcoin address is derived from an ECDSA public key by repeatedly hashing the public key with the SHA-256 and RIPEMD-160 hash functions. Any party who learns the ECDSA private key corresponding to an address can create a transaction moving any funds associated with that address to an account that they control, and authenticate the transaction by creating a valid signature with the private key.

2.2 ECDSA

In the ECDSA signature scheme, public global parameters include an elliptic curve E together with a generator point G that generates a group of order n on E. A signer's private key is an integer d modulo n, and the corresponding public key is the point $Q = dG$. The security of the scheme rests on the presumed difficulty of the elliptic curve discrete log problem (finding d given only G and Q over certain elliptic curves).

To sign a message with hash h, the signer generates an integer nonce k modulo n. The signature is the pair (r, s), where r is the x coordinate of kG and

$$s = k^{-1}(h + dr) \bmod n. \tag{1}$$

To verify the signature, the recipient tests for equality between r and the x coordinate of the point $(hs^{-1})G + (rs^{-1})Q$.

Nonce generation. While initial descriptions of the ECDSA algorithm specified that the nonce k should be generated at random, there has been a long history of catastrophic vulnerabilities caused by random number generation failures. A single compromised signature nonce k reveals the secret key d by solving Equation 1 for this value. Repeated (EC)DSA nonces used to sign distinct messages with the same key allow straightforward recovery of the private key by solving the two relations given by Equation 1 for each signature for the two unknowns d and k. Reused nonces have been observed repeatedly since 2013 on the Bitcoin blockchain in academic works [4,7,6,5], and there appear to be systematic thefts from keys compromised in this way[3].

Modern ECDSA implementations (including Bitcoin core, Ethereum, and other cryptocurrencies) generate *deterministic* nonces, for example following RFC 6979 and instantiating HMAC_DRBG with the secret key d and the hash h. Nevertheless, use of such an algorithm is impossible to enforce, and observations from prior work demonstrate that custom implementations still use a variety of methods to generate nonces [5].

Signature normalization. The signatures (r, s) and $(r, -s)$ are both valid for the same message given the above scheme. To ensure that signatures are unique, Bitcoin and other cryptocurrencies use the convention that only the smaller of the values $-s$ and s is valid. This has the effect of negating the nonce k for some signatures.

2.3 Lattice problems and algorithms

A lattice is a discrete additive subgroup of $\mathbb{R}^m$. Explicitly, a lattice $L(B)$ is generated from integer linear combinations of a set of basis vectors $B = \{b_1, \ldots b_m\}$ with $b_i \in \mathbb{R}^m$. The problem of computing the shortest vector (SVP) in a lattice given an arbitrary basis is NP-hard, and hard to approximate for constant factors.

The Lenstra Lenstra Lovász (LLL) lattice reduction algorithm [11] gives an exponential approximation for the shortest vector in polynomial time; the Block Korkine-Zolotarev (BKZ) algorithm [13,14] can be used to solve exact SVP albeit in exponential time; more generally one can interpolate between these extremes to achieve intermediate approximation factors by adjusting the BKZ block size.

2.4 Hidden number problem

Boneh and Venkatesan formulated the hidden number problem to study hard-core bits for Diffie-Hellman [3]. In the *hidden number problem*, the goal is to reconstruct a "hidden number" α given only the top bits a_i of samples $t_i \cdot \alpha \mod p$ (where t_i is also known). Explicitly, we are searching for α satisfying the equations

$$t_i \cdot \alpha = a_i + b_i \bmod p$$

[3] https://bitcointalk.org/index.php?topic=581411.msg9809990#msg9809990 posted December 11, 2014 by Jochen Hoenicke (johoe), retrieved March 2, 2023

with some fixed bound B on the (unknown) lower bits such that $b_i < B < p$. In Boneh and Venkatesan's work introducing the hidden number problem, they construct a lattice basis using the sample data and solve the CVP (closest vector problem) on that lattice. It is standard in practice to use Kannan's embedding to use an SVP algorithm to solve the problem instead [2,1]. In this formulation the lattice is generated by the rows of

$$B = \begin{bmatrix} p & & & & & \\ & p & & & & \\ & & \ddots & & & \\ & & & p & & \\ t_1 & t_2 & \dots & t_m & B/p & \\ a_1 & a_2 & \dots & a_m & & B \end{bmatrix}$$

Most presentations hope that the target vector $v_t = (b_1, b_2, \dots b_m, B\alpha/p, B)$ is short enough to be found by an algorithm like BKZ that solves SVP; this vector can then be used to construct α. We expect this algorithm to succeed when the ℓ_2 norm of v_t is less than the Gaussian Heuristic $\mathrm{gh}(\cdot)$ for $L(B)$, which gives the expected length of the shortest vector of a random lattice. This is approximately $\mathrm{gh}(L(B)) \approx \sqrt{\dim L/(2\pi e)} \det L(B)^{1/\dim L}$ (where $L = L(B)$). It is possible to use sieving and enumeration techniques to increase the probability of success for this approach after applying the optimizations below [1].

2.5 ECDSA as a Hidden Number Problem

Howgrave-Graham and Smart [9] and Nguyen and Shparlinski [12] showed that the above lattice-based algorithm for solving the hidden number problem could be used to recover an (EC)DSA secret key from most significant bits of nonces from many signatures.

In an implementation of ECDSA that leaks the most significant bits of the nonces k_i, we can recover the secret key d as follows. Given signatures (r_i, s_i) and corresponding hashes h_i, each signature satisfies the equation

$$-s_i h_i + k_i = s_i^{-1} r_i \cdot d \bmod n$$

This gives an instance of the hidden number problem with $\alpha = d$, $a_i = -s_i h_i$, $t_i = s_i^{-1} r_i$, $b_i = k_i$, and $p = n$. Thus, with enough signatures, we can construct the lattice basis B as above and recover d.

Lattice attacks on applications of ECDSA. The above algorithm is commonly used for ECDSA cryptanalysis in the context of side-channel attacks [2,10,8,16]. Breitner and Heninger [5] applied the lattice-based algorithm for the hidden number problem to compute ECDSA private keys in cryptocurrencies from signatures on the blockchain that had been generated with poorly generated nonces. These attacks all required multiple signatures with vulnerable nonces to have been generated from a given key to enable key recovery; in the most common

case observed in the wild, it was possible to recover a private signing key from two signatures whose nonces were shorter than 128 bits. Since it is not possible to tell in advance which signatures might be vulnerable, an attacker searching for vulnerable signatures may need to test all pairs of signatures from each key.

No general attack of this form was possible for keys associated with only one signature.

3 Half Nonce Attack

The starting point for our attack is the observation of signatures in the Bitcoin blockchain whose nonces appear to have been generated by concatenating the high bits of h together with the high bits of d. That is, the nonce k satisfies

$$k = 2^\ell h_{\text{msb}} + d_{\text{msb}} \tag{2}$$

where ℓ is the length of d_{msb} (in the context of Bitcoin, $\ell = 128$).

Fifty four of these signature nonces appear in the data of Breitner and Heninger [5], who noted having recovered signature nonces that shared least significant bits of the nonce with d, but apparently did not notice the shared bits with the hash h. Their attack required multiple signatures with nonces of this form to recover the private key, and testing all possible combinations of signatures for keys that had generated large numbers of signatures on the blockchain was prohibitively expensive.

We observe that it is possible to recover the private key from a signature generated with a nonce of this form *from a single signature* using a lattice-based attack.

3.1 Setup and Main Attack

From Equation 2 defining k and Equation 1 defining s, we can derive the equation

$$\underbrace{(2^\ell - sr^{-1})}_{A} d_{\text{msb}} + d_{\text{lsb}} + \underbrace{(h - 2^\ell s h_{\text{msb}})r^{-1}}_{b} = 0 \bmod n \tag{3}$$

This is an affine equation in d_{msb} and d_{lsb} over $\mathbb{Z}_n$, where we can expect $d_{\text{msb}}, d_{\text{lsb}}$ to both be smaller than 2^ℓ, and $2\ell \leq \lg n$. The lattice generated by the rows of the basis

$$B = \begin{bmatrix} n & 0 & 0 \\ A & 1 & 0 \\ b & 0 & 2^\ell \end{bmatrix} \tag{4}$$

will contain the target vector $v_t = (d_{\text{lsb}}, d_{\text{msb}}, 2^\ell)$ by construction. We have $\det L(B) = \det B = n2^\ell$, $\dim L = 3$, and $|v_t|_2 \leq \sqrt{3}2^\ell$. The Gaussian Heuristic criteria is very close to but not quite satisfied since we need $\sqrt{3}2^\ell < \sqrt{3/(2\pi e)}2^\ell$. Thus we expect the target vector to be the closest vector in the lattice only a fraction of the time. Empirically, without further optimizations, this lattice construction succeeded with probability 27.1%.

We can increase the success probability by applying the optimizations below, and brute forcing a few bits or using sieving or enumeration with predicate techniques [1] to find the target vector.

3.2 Optimizations

In order to increase our success probability, we apply two optimizations.

Recentering. The recentering optimization observes that the variables d_{msb} and d_{lsb} are always positive, while lattice vectors can take positive or negative values. We define the variables $d'_{\text{msb}} = d_{\text{msb}} - 2^{\ell-1}$ and $d'_{\text{lsb}} = d_{\text{lsb}} - 2^{\ell-1}$ which recenter the values of the unknowns around zero, and then set the bottom right entry of B in Equation 4 to $2^{\ell-1}$ and redefine A and b in Equation 3 accordingly. Then, we expect the shortest vector in the lattice to take the form $(d'_{\text{lsb}}, d'_{\text{msb}}, 2^{\ell-1})$ (up to sign). This one-bit change significantly improves the success rate of the algorithm at nearly no cost to running time: with this optimization alone we achieve a success probability of 76.2%.

Brute forcing. The success rate can also be improved by brute forcing the top few bits of d_{lsb} and d_{msb} to make the unknown values smaller, at a cost of having to run 2^t instances when t bits are brute forced. We ended up brute forcing four bits total, split evenly between d_{msb} and d_{lsb}. This number was chosen as a practical trade-off between running time and success rate.

Since nonces can be negated by the signature normalization process, we also brute force both positive and negative values for the resulting nonces, resulting in $2 \cdot 2^4 = 32$ total lattice reductions for each signature.

Applying both recentering and brute forcing, our algorithm achieved a 99.6% success rate on synthetic data. At this success rate, the algorithm is likely to detect almost all vulnerable signatures when run on the entire Bitcoin blockchain.

Sieving with predicate. Using the Sieving with Predicate algorithm from [1] with recentering and no brute forcing, we were able to achieve 99.99% success rate on synthetic data, at an even faster average rate of 0.48 seconds per signature per core on our real data. This also allowed us to recover an additional 440 signatures, including signatures from August 2022 that suggest that the attacker below is still active. However, these extra signatures are not included in the analysis below due to time constraints.

4 Implementation

We implemented the algorithm in Sage [15] using the above optimizations and G6K's implementation of the BKZ algorithm. A single signature takes 0.688 seconds to test on an Intel Xeon E5-2699A v4 CPU core.

To collect the dataset of 2.14 billion signatures from the Bitcoin blockchain, we instrumented the signature verification code in the official Bitcoin client to

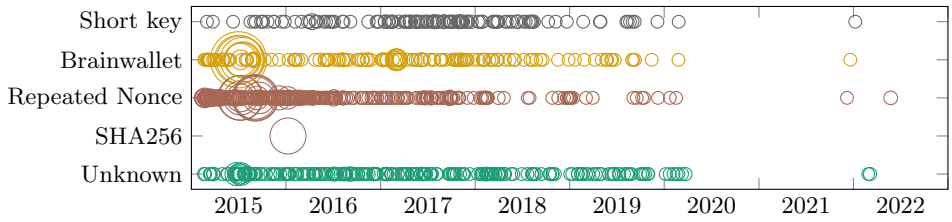

Fig. 1. Compromised signature classification. We plot signatures with compromised nonces over time, grouped by the likely reason the source address is known to be compromised. Larger circles correspond to more transaction inputs on a given date.

write the data from each signature (message hash, signature, public key and transaction id) to a separate file, and re-validated the full Bitcoin blockchain, up to block height 738173 (May 2022).

We then tested each signature using our attack. Since we can verify success using the public key, there are no false positives. Each lattice reduction only requires a single signature, so the attack is perfectly parallellizable; we ran the algorithm on around 1000 physical cores using Slurm [17]. A single core was able to test 120,000 signatures per day. The computation completed in roughly 49 cpu-years, or 18 calendar days.

5 Analysis

We found 88,230 vulnerable signatures from 873 unique secret keys. These were part of 7,242 transactions. Of these, 5,010 had multiple inputs. The transactions transfer funds from 893 unique input addresses[4] to 53 unique output addresses, and 37 addresses occur both as input and as output.

We found a total of 222 Bitcoin that were moved in these transactions. Of these, 55 Bitcoin were moved to an address that is *not* also an input address in these transactions. None of the compromised addresses had any funds as of this writing.

We observe compromised signatures between 2015-02-14 and 2022-05-24. The plot in Figure 1 shows very high activity in 2015, a break from 2020 to 2021, and a few occurrences again in 2022 before we stopped collecting data.

5.1 Source address analysis

Further inspection of the 893 source addresses shows that most of these were known to be compromised prior to our work. We were able to identify the following vulnerable categories:

[4] There can be multiple addresses formed from a single private key.

– **Short keys:** some addresses belong to private keys that are short ($d < 2^{128}$). For example the address 1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZm corresponds to a private key of $d = 1$.
– **Brainwallet:** A brainwallet is a way to generate a private key from a passphrase that one can remember. This makes such Bitcoin addresses susceptible to brute-force attacks, and there exist databases of Bitcoin addresses that arise from weak or compromised brainwallet keyphrases[5].
– **Repeated nonces:** As explained above, re-using a nonce in a signature makes key recovery quite simple. Jochen Hoenicke publishes a list of addresses leaked that way,[6] which we use to identify these addresses.
– **SHA256 inversion hash:** We found one non-ECDSA address, 3GfLKx6ius6MwUetY6gAqeabvoUZJ2qheQ, among the inputs to transactions that contained vulnerable nonces. This address is protected by a simple hash: Its "pkscript" allows anyone who knows the preimage to a certain SHA256 hash to withdraw funds from this address, revealing the preimage in the process.

We were able to attribute most of the source addresses associated with these vulnerable transactions into at least one of these categories of compromised addresses. See Table 1. Many of the remaining unclassified addresses appear on the web, for example as example addresses in Bitcoin-related libraries or tools. It is well known that bots sweep funds from such compromised addresses on the blockchain. Figure 2 shows the flow of Bitcoin through these transactions.

Short key	96
Brainwallet	89
Repeated nonce	512
SHA256	1
Unknown	195
Total	893

Table 1. Source addresses

5.2 Attribution

The unusual prevalence of well-publicized vulnerabilities that can be exploited to reveal the private keys corresponding to the source addresses of these transactions leads us to the hypothesis that these transactions represent an attacker stealing funds from the original owners of these addresses. The quirky signature nonce construction that has revealed the source address private keys to *us* may be an implementation artifact of custom-written attack script that this attacker is using to snatch funds from addresses with compromised private keys, as well as to transfer funds between intermediary addresses.

We have found evidence linking this activity to the person behind the pseudonym "amaclin" on https://bitcointalk.org and https://bitcoin.stackexchange.com.

– Amaclin is said to have "a long history of scamming other users via double-spends and other technical blockchain tricks, and even ground Bitcoin itself

[5] We used https://eli5.eu/brainwallet/, which contains 18,982 addresses, but is incomplete.
[6] https://johoe.mooo.com/bitcoin/broken.txt

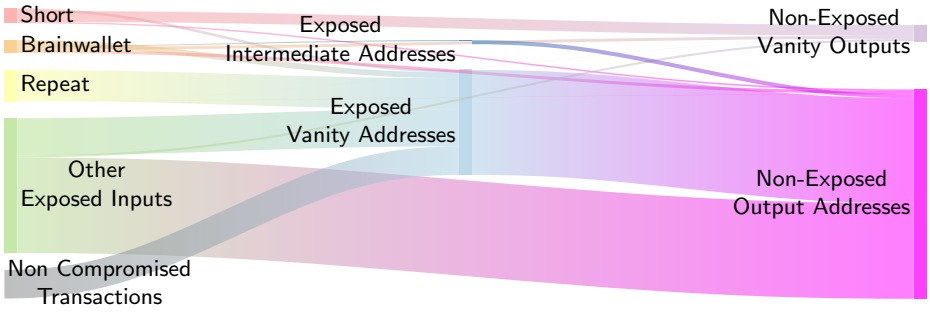

Fig. 2. Flow of Bitcoin through Compromised Transactions. We were able to recover the private keys for every input address in every transaction that contained a vulnerable signature. The vulnerable transactions moved funds from 893 compromised input addresses, through a set of 37 compromised intermediate addresses (most of which were vanity addresses following a recognizable pattern), to 16 output addresses whose keys have not been exposed through vulnerable signatures.

to a near halt with blockchain malleability attacks,"[7] which fits the observations in the previous section. Amaclin was interviewed about the mentioned malleability attack by bitcoinmagazine.com[8] and vice.com[9].

– 32 of the destination addresses of transactions with vulnerable signatures are vanity addresses starting with `1aa` or `1xy`, and there are indications that these amaclin creates and uses such addresses, and that these addresses are used in draining compromised accounts:

  - For example, transaction 77dd8a24288aa87a7976adeff0579d60ac50b384b8a28bd589a47459573c9345 from 30 December 2014 (which is not among those with a vulnerable signatures) has among its 17 input addresses 15 such vanity addresses, but also 1ENnzep2ivWYqXjAodTueiZscT6kunAyYs, which amaclin put on their StackExchange profile page[10].
  - In the aftermath of a reused-nonce incident on blockchain.info[11], Jochen Hoenicke writes[12] that amaclin is among those grabbing funds, and that `1aa` and `1xy` addresses were used (without explicitly linking these two).

[7] https://bitcointalk.org/index.php?topic=5154360.0, posted June 14, 2019 by eddie13, retrieved March 3, 2023

[8] https://bitcoinmagazine.com/culture/the-who-what-why-and-how-of-the-ongoing-transaction-malleability-attack-1444253 Oct 7, 2015, retrieved March 2, 2023

[9] https://www.vice.com/en/article/pga7m9/i-broke-bitcoin, October 7, 2015, retrieved March 2, 2023

[10] https://bitcoin.stackexchange.com/users/12983/amaclin

[11] https://web.archive.org/web/20141225032628/http://blog.blockchain.com/2014/12/08/blockchain-info-security-disclosure/, posted December 8, 2014

[12] https://bitcointalk.org/index.php?topic=581411.msg9888800#msg9888800, posted December 19, 2014, retrieved March 2, 2023

- Amaclin claims on September 14, 2015 that address 1aa5cmqmvQq8YQT EqcTmW7dfBNuFwgdCD is his[13], used to collect "dust" while a "game" is going on. Our analysis finds this address as a destination in a transaction with vulnerable signatures. Assuming amaclin is the only owner of this address, this shows that they are creating these peculiar signatures.

– In a post on Sept 16, 2015[14], he further explains various tricks (e.g. addresses based on hash inversion, see Section 5.1) to make short signatures and build transactions that sweep compromised accounts before others. He writes:
"Yes, you need to make the miners prefer my transaction over the original one. To do this, you need to either give more commissions or make the transaction smaller. In short, so that the commission-per-kilobyte would be maximum. This trick allows you to reduce the length of the signature. It cannot always be used for signing because the private key becomes known. But in this case, I don't care that someone recognizes the key. It was already published, so whether it will be known not by a thousand people, but by three thousand - it does not matter. It is important to have time to transfer the loot to your address before others."

The connections between the vulnerable signatures and addresses that can be attributed to amaclin lead us to the hypothesis that code written and used by them is creating the pecular signatures. We reached out to this individual by email in October 2022 and did not receive a response. So far, we have not seen evidence of another source for transactions with vulnerable signatures.

6 Conclusion

This attack has a simple countermeasure: deterministic ECDSA nonce generation as described in, for example, RFC 6979, which is implemented in the core library for Bitcoin, Ethereum, and other cryptocurrencies.

Our much more powerful and exact lattice attack has allowed us to identify 90,000 vulnerable signatures in the Bitcoin blockchain by exploiting this previously unnoticed vulnerable nonce pattern, where the methods of Breitner and Heninger were only able to identify 54 signatures of this form.

Our analysis also illustrates how vulnerabilities in a custom ECDSA implementation can not only reveal private keys, but can also be used to compromise Bitcoin-pseudonymity and addresses to (online) personas.

This is also a nice example of a cryptographic implementation mistake leading to interesting cryptanalysis: our attack easily extends to other nonce constructions from contiguous chunks of d.

[13] https://bitcointalk.org/index.php?topic=1179542.msg12417028#msg12417028, retrieved March 3, 2023

[14] https://bitcointalk.org/index.php?topic=1179542.msg12434341#msg12434341 retrieved March 3, 2023. Translation from Russian using Google Translate

Acknowledgements

We are grateful to Jochen Hoenicke for helpful discussions.

References

1. Albrecht, M.R., Heninger, N.: On bounded distance decoding with predicate: Breaking the "lattice barrier" for the hidden number problem. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 528–558. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-77870-5_19
2. Benger, N., van de Pol, J., Smart, N.P., Yarom, Y.: "ooh aah... just a little bit": A small amount of side channel can go a long way. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 75–92. Springer, Heidelberg (Sep 2014). https://doi.org/10.1007/978-3-662-44709-3_5
3. Boneh, D., Venkatesan, R.: Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 129–142. Springer, Heidelberg (Aug 1996). https://doi.org/10.1007/3-540-68697-5_11
4. Bos, J.W., Halderman, J.A., Heninger, N., Moore, J., Naehrig, M., Wustrow, E.: Elliptic curve cryptography in practice. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 157–175. Springer, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-662-45472-5_11
5. Breitner, J., Heninger, N.: Biased nonce sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 3–20. Springer, Heidelberg (Feb 2019). https://doi.org/10.1007/978-3-030-32101-7_1
6. Brengel, M., Rossow, C.: Identifying key leakage of bitcoin users. In: Bailey, M., Holz, T., Stamatogiannakis, M., Ioannidis, S. (eds.) Research in Attacks, Intrusions, and Defenses. pp. 623–643. Springer International Publishing, Cham (2018)
7. Courtois, N.T., Emirdag, P., Valsorda, F.: Private key recovery combination attacks: On extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor RNG events. Cryptology ePrint Archive, Report 2014/848 (2014), https://eprint.iacr.org/2014/848
8. Genkin, D., Nissan, N., Schuster, R., Tromer, E.: Lend me your ear: Passive remote physical side channels on PCs. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 4437–4454. USENIX Association, Boston, MA (Aug 2022), https://www.usenix.org/conference/usenixsecurity22/presentation/genkin
9. Howgrave-Graham, N.A., Smart, N.P.: Lattice attacks on digital signature schemes. Designs, Codes and Cryptography **23**(3), 283–290 (Aug 2001). https://doi.org/10.1023/A:1011214926272, https://doi.org/10.1023/A:1011214926272
10. Jancar, J., Sedlacek, V., Svenda, P., Sys, M.: Minerva: The curse of ECDSA nonces. IACR TCHES **2020**(4), 281–308 (2020). https://doi.org/10.13154/tches.v2020.i4.281-308, https://tches.iacr.org/index.php/TCHES/article/view/8684
11. Lenstra, A.K., Lenstra, H.W., Lovasz, L.: Factoring polynomials with rational coefficients. MATH. ANN **261**, 515–534 (1982)
12. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the elliptic curve digital signature algorithm with partially known nonces. Designs, Codes and Cryptography **30**(2), 201–217 (Sep 2003). https://doi.org/10.1023/A:1025436905711, https://doi.org/10.1023/A:1025436905711

13. Schnorr, C.P.: A hierarchy of polynomial time lattice basis reduction algorithms. Theor. Comput. Sci. **53**(2-3), 201–224 (Aug 1987). https://doi.org/10.1016/0304-3975(87)90064-8, http://dx.doi.org/10.1016/0304-3975(87)90064-8
14. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Math. Program. **66**(2), 181–199 (Sep 1994). https://doi.org/10.1007/BF01581144, http://dx.doi.org/10.1007/BF01581144
15. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 8.7) (2022), `https://www.sagemath.org`
16. Weiser, S., Schrammel, D., Bodner, L., Spreitzer, R.: Big numbers - big troubles: Systematically analyzing nonce leakage in (EC)DSA implementations. In: Capkun, S., Roesner, F. (eds.) USENIX Security 2020. pp. 1767–1784. USENIX Association (Aug 2020)
17. Yoo, A.B., Jette, M.A., Grondona, M.: Slurm: Simple linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) Job Scheduling Strategies for Parallel Processing. pp. 44–60. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)