NTNU | Norwegian University of Science and Technology

RANDOMNESS 2: BREAKING ECDSA

TTM4205 – Lecture 4

Tjerand Silde

25.08.2026

Contents

Contents

Reference Material

These slides are based on:

- JPA: parts of chapter 12
- DW: parts of chapter 7
- Papers listed in the slides

Contents

Elliptic Curves

- Let $\mathbb{F}_p$ be a finite field of prime order $p > 3$.
- Let $\mathcal{O}$ be the point at infinity, the neutral element.

$$E_{a,b} = \left\{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p \quad | \quad y^2 = x^3 + ax + b\right\} \cup \{\mathcal{O}\}$$

- We need $4a^3 + 27b^2 \neq 0$, so that the curve is non-singular.
- The points then form an Abelian group under the addition below.

Elliptic Curves

Adding two points together: $R = P + Q$

- Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points on the elliptic curve $E_{a,b}$.
- If $P = \mathcal{O}$, then $P + Q = Q$, and if $Q = \mathcal{O}$, then $P + Q = P$.
- If $x_1 = x_2$ and $y_1 = -y_2$, then $P + Q = \mathcal{O}$.
- Otherwise, let $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = \lambda(x_1 - x_3) - y_1$, where

$$\lambda = \begin{cases} \frac{3x_1^2 + a}{2y_1} & \text{if } P = Q \\ \frac{y_1 - y_2}{x_1 - x_2} & \text{otherwise,} \end{cases}$$

and output $R = (x_3, y_3)$. All arithmetic is in $\mathbb{F}_p$.

Elliptic Curves

Scalar multiplication: $Q = x \cdot P$

$$x \cdot P = \underbrace{P + P + \cdots + P}_{x \text{ times}}$$

- Adding P to itself x times takes x operations: far too slow for a 256-bit scalar. Can we speed this up?

Elliptic Curves

Double-and-Add

- Write the scalar in binary: $x = (x_{d-1} \dots x_0)_2$, where $d = \lceil \log_2 x \rceil$.
- Set $R \leftarrow \mathcal{O}$. For $i = d - 1$ down to 0: $R \leftarrow 2R$, and if $x_i = 1$ then $R \leftarrow R + P$.
- Output $R = x \cdot P$: at most $2 \log_2 x$ doublings/additions, not x additions.

Why Elliptic Curves?

The Discrete Logarithm Problem (DLP)

- Let $\mathbb{G}_p$ be a group of prime order p with generator g.
- Let $h = g^x$ in $\mathbb{G}_p = \mathbb{F}_p^*$ for some integer $x \in \mathbb{Z}_p$.
- The DLP is to find x given g and h, that is, $x = \log_g h$.
- In the multiplicative group $\mathbb{F}_p^*$ this is sub-exponential using index calculus and the number field sieve, which is why p has to be around 3072 bits.

Why Elliptic Curves?

The Elliptic Curve Discrete Logarithm Problem (ECDLP)

- Let $E_{a,b}$ be an elliptic curve of order n over a finite field $\mathbb{F}_p$.
- Let $G \in E_{a,b}$ be a generator and let $Q = x \cdot G$ for some $x \in \mathbb{Z}_n$.
- The ECDLP is to find x given G and Q on the elliptic curve $E_{a,b}$.
- The ECDLP is believed to be hard, so we can build secure protocols, and the security of ECDH, ECDSA and other schemes rests on it.
- No index calculus is known here: the best algorithms are generic, in time $\approx \sqrt{n}$, so 256-bit curves give 128-bit security.

Contents

ECDSA Signature Algorithm

- **Keys:** $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_n^*$ and $\mathsf{pk} = \mathsf{sk} \cdot G$, for a generator G of prime order n.
- **To sign** m**:** compute the hash $h \leftarrow \mathrm{H}(m)$, and sample a nonce $k \leftarrow\!\!\$\ \mathbb{Z}_n^*$.
- Compute the nonce point $R = (x, y) \leftarrow k \cdot G$, then $r \leftarrow x \bmod n$ and $s \leftarrow k^{-1}(h + r \cdot \mathsf{sk}) \bmod n$, starting over if $r = 0$ or $s = 0$.
- **Output:** the signature $\sigma = (r, s)$ on the message m.
- The nonce k must be fresh, uniform, and secret. What happens if it is not?

ECDSA Signature Algorithm

Reused randomness

- If k is reused for two different messages, the secret key can be recovered.
- Reusing k reuses r, since r depends only on k: $s_i = k^{-1}(h_i + r \cdot \mathrm{sk}) \bmod n$.

$$
\begin{aligned}
s_1 - s_2 &= k^{-1}(h_1 + r \cdot \mathrm{sk}) - k^{-1}(h_2 + r \cdot \mathrm{sk}) = k^{-1}(h_1 - h_2) \\
k &= (h_1 - h_2)(s_1 - s_2)^{-1} \bmod n \\
\mathrm{sk} &= (s_1 \cdot k - h_1)\, r^{-1} \bmod n
\end{aligned}
$$

- The hashes differ, so $s_1 \neq s_2$, and n is prime: both divisions are defined.

Nonce Failures in the Wild

- **Sony PlayStation 3 (2010).** The same constant k for every signature. Two signed binaries gave the console signing key.
- **Android Bitcoin wallets (2013).** A flawed `SecureRandom` repeated nonces, and wallets were emptied.
- **Partial leakage.** A few bits per nonce suffice: Minerva and TPM-FAIL leak the bit-length of k by timing, LadderLeak needs less than one bit.

ECDSA Signature Verification

- **Input:** Message m, signature (r, s), and the public key pk.
- **Output:** Accept or reject:
  - Reject if $r, s \notin \mathbb{Z}_n^*$, or if pk is invalid ($\mathsf{pk} = \mathcal{O}$ or $\mathsf{pk} \notin E_{a,b}$).
  - Compute $h \leftarrow \mathrm{H}(m)$, then $u_1 \leftarrow h \cdot s^{-1} \bmod n$ and $u_2 \leftarrow r \cdot s^{-1} \bmod n$.
  - Compute the point $(x, y) \leftarrow u_1 \cdot G + u_2 \cdot \mathsf{pk}$, and reject if it is $\mathcal{O}$.
  - Accept the signature if $r \equiv x \pmod{n}$, else reject.
- It works because $u_1 \cdot G + u_2 \cdot \mathsf{pk} = s^{-1}(h + r \cdot \mathsf{sk})G = k \cdot G = R$.

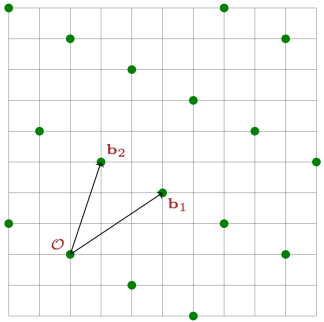
Contents

Lattices

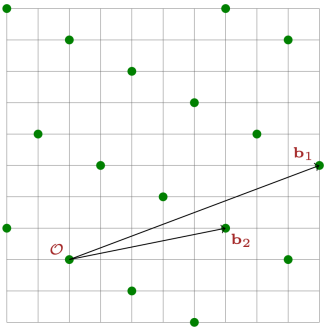
- Let $\mathsf{B} = [\mathsf{b}_1, \ldots, \mathsf{b}_k] \in \mathbb{R}^{d \times k}$ be a linearly independent set in $\mathbb{R}^d$.
- A lattice, $\Lambda(\mathsf{B})$, generated by the matrix B is the set of all linear combinations of the columns of B with *integer* coefficients.
- B is called a basis for $\Lambda(\mathsf{B})$, and every lattice has infinitely many bases: a few short and nearly orthogonal, but most are long and skewed.

$$\Lambda(\mathsf{B}) = \left\{ \mathsf{B}\mathsf{x} : \mathsf{x} \in \mathbb{Z}^k \right\} = \left\{ \sum_{i=1}^{k} x_i \cdot \mathsf{b}_i : x_i \in \mathbb{Z} \right\}$$

Lattices (intuition)



A short, nearly orthogonal basis



A long, skewed basis

The *same* lattice, but two different bases: the green points are identical.

Lattice Problems

Definition (Shortest Vector Problem.)

Given a lattice Λ, find a vector $\mathbf{v} \in \Lambda \setminus \{0\}$ such that $\|\mathbf{v}\| \leq \|\mathbf{u}\|$ for all $\mathbf{u} \in \Lambda \setminus \{0\}$. Its length is written $\lambda_1(\Lambda)$.

Definition (Closest Vector Problem.)

Given a lattice Λ and a target $\mathbf{u}$ that need not lie in Λ, find the lattice vector $\mathbf{v}$ such that $\|\mathbf{u} - \mathbf{v}\| \leq \|\mathbf{u} - \mathbf{v}'\|$ for all $\mathbf{v}' \in \Lambda$.

Solving Lattice Problems

Gram-Schmidt Orthogonalization

- **Input:** Basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_k)$ of the lattice Λ.
- **Goal:** Compute orthogonal basis $\tilde{\mathbf{B}} = (\tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_k)$ and coefficients $\mu_{i,j}$.

$$\tilde{\mathbf{b}}_1 = \mathbf{b}_1, \qquad \tilde{\mathbf{b}}_i = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\, \tilde{\mathbf{b}}_j \quad (i \geq 2), \qquad \mu_{i,j} = \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2}\,.$$

- **Intuition:** Project $\mathbf{b}_i$ onto the span of the previous $\tilde{\mathbf{b}}_j$ and subtract; what is left is orthogonal to all of them. This should be known from linear algebra.

Solving Lattice Problems

The Lenstra–Lenstra–Lovász (LLL) algorithm

- **Input:** A basis $\mathrm{B} = (\mathrm{b}_1, \ldots, \mathrm{b}_k)$ for a lattice Λ.
- **Output:** A **reduced basis** $\mathrm{B}' = (\mathrm{b}'_1, \ldots, \mathrm{b}'_k)$ for Λ.
- **GSO:** Compute the Gram-Schmidt basis as above: $\tilde{\mathrm{B}} \leftarrow \mathrm{GSO}(\mathrm{B})$.
- **Size reduction:** for all i, j, if $\left|\mu_{i,j}\right| > \frac{1}{2}$, then update $\mathrm{b}_i \leftarrow \mathrm{b}_i - \lfloor \mu_{i,j} \rceil \, \mathrm{b}_j$.
- **Lovász condition:** for all i, check $\|\tilde{\mathrm{b}}_i\|^2 \geq (\delta - \mu_{i,i-1}^2) \, \|\tilde{\mathrm{b}}_{i-1}\|^2$, for $\delta \in (1/4, 1)$.
- If it is violated, swap $\mathrm{b}_i \leftrightarrow \mathrm{b}_{i-1}$, recompute the GSO, and start over.
- **Return** the reduced basis B' with the guarantee $\|\mathrm{b}'_1\| \leq 2^{(k-1)/2} \lambda_1(\Lambda)$.

How Does This Help Us?

- If we are fortunate, the first vector $\mathfrak{b}_1'$ of the reduced basis is the shortest vector in the lattice.
- It is in any case far shorter than the vector we started with, and in the low dimensions we work with here it usually is the shortest one.
- Why does this help? Consider what happens when ECDSA is implemented so that the nonces are smaller than they are supposed to be.

Short Randomness

Prerequisites

We have ℓ signatures of the form

$$s_i \equiv k_i^{-1}(h_i + r_i \cdot \mathrm{sk}) \mod n, \quad \text{for } i \in [\ell]$$

If the nonce is "too short", then its most significant bits are all zero.

Suppose $\ell = 3$ signatures are made with 128-bit nonces, for example from a truncated PRNG output, while a 256-bit curve calls for 256-bit nonces.

What can we do?

In general, each nonce k_i can then be written in the form

$$k_i = 2^{128}a + b_i, \quad b_i < 2^{128}$$

Short Randomness: What Can We Do?

$$
\begin{aligned}
s_i &\equiv k_i^{-1}(h_i + r_i \cdot \mathsf{sk}) \mod n \\
s_i &\equiv (2^{128}a + b_i)^{-1}(h_i + r_i \cdot \mathsf{sk}) \mod n \\
b_i + 2^{128}a &\equiv s_i^{-1}(h_i + r_i \cdot \mathsf{sk}) \mod n \\
b_i + 2^{128}a &\equiv s_i^{-1}h_i + s_i^{-1}r_i \cdot \mathsf{sk} \mod n
\end{aligned}
$$

We now have an equation describing the nonce k_i in terms of public values and the unknowns sk, a, and b_i. But how do we actually use it?

- First, we know that $a = 0$ because the nonce is short, so the unknown b_i is bounded by $B = 2^{128}$ while $n \approx 2^{256}$.

$$b_i \equiv s_i^{-1}h_i + s_i^{-1}r_i \cdot \mathsf{sk} \mod n$$

Answer: The Hidden Number Problem

- Our problem: we want to recover an unknown scalar sk, knowing only partial information about multiples of it. This is the Hidden Number Problem of Boneh and Venkatesan (1996).
- What we know: some partial information about each fresh nonce k_i.
- We also know the values s_i, r_i, h_i for each signature and the modulus n.
- So we reformulate: let $t_i = s_i^{-1} r_i$ and $u_i = s_i^{-1} h_i$, both computable from the signature and the message alone. With $a = 0$ we then get

$$b_i \equiv t_i \cdot \mathrm{sk} + u_i \mod n$$

Formalizing the Hidden Number Problem (HNP)

The adversary is given ℓ pairs of integers $\{(t_i, u_i)\}_{i=1}^{\ell}$ such that $t_i \cdot \mathsf{sk} + u_i \bmod n = b_i$, where $|b_i| < B$ for a bound $B < n$. (1)

Solving the Hidden Number Problem

Let us set up the problem as a system of linear equations, assuming each b_i is 128 bits long (128 zero bits precede it in k_i), with ℓ signatures:

$$\begin{aligned} b_1 &\equiv t_1 \cdot \mathsf{sk} + u_1 \mod n \\ b_2 &\equiv t_2 \cdot \mathsf{sk} + u_2 \mod n \\ &\vdots \\ b_\ell &\equiv t_\ell \cdot \mathsf{sk} + u_\ell \mod n \end{aligned}$$

The b_i are short while everything else is the size of n. This is an instance of the shortest vector problem, which we can solve using the LLL algorithm. As a rule of thumb, if w bits of each nonce are known, then about $\lceil \log_2 n/w \rceil$ signatures make the target vector short enough to be found.

Solving the Hidden Number Problem

Our goal is now to construct a lattice whose shortest vector is our solution. Written as a matrix equation, the system becomes:

$$\begin{bmatrix} j_1 & j_2 & j_3 & \mathsf{sk} & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 & 0 \\ 0 & n & 0 & 0 & 0 \\ 0 & 0 & n & 0 & 0 \\ t_1 & t_2 & t_3 & B/n & 0 \\ u_1 & u_2 & u_3 & 0 & B \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & b_3 & \mathsf{sk} \cdot B/n & B \end{bmatrix}$$

The unknown integers j_i absorb the reductions modulo n, which is what the first three rows are for, and the last two columns lift sk and the constant into the lattice, scaled so that they do not dominate the norm.

The b_i are short, so the target vector is far shorter than a typical lattice vector, and LLL returns it first. Its second-to-last coordinate is $\mathsf{sk} \cdot B/n$, from which the secret key follows immediately.

Partially Equal Randomness

But what if $a \neq 0$? The nonces are now full length, but a broken PRF makes them share their most significant half, and we do not know that shared part. Can we still recover sk?

Partially Equal Randomness

Recall:

$$s_i \equiv k_i^{-1}(h_i + r_i \cdot \mathrm{sk}) \mod n$$

and

$$k_i = 2^{128}a + b_i, \quad b_i < 2^{128}, a \neq 0$$

$$2^{128}a + b_i \equiv s_i^{-1}h_i + s_i^{-1}r_i \cdot \mathrm{sk} \mod n$$

Partially Equal Randomness

Recall the equations describing $k_i = 2^{128}a + b_i$:

$$
\begin{aligned}
2^{128}a + b_1 &\equiv s_1^{-1}h_1 + s_1^{-1}r_1 \cdot \mathrm{sk} \mod n \\
2^{128}a + b_2 &\equiv s_2^{-1}h_2 + s_2^{-1}r_2 \cdot \mathrm{sk} \mod n \\
2^{128}a + b_3 &\equiv s_3^{-1}h_3 + s_3^{-1}r_3 \cdot \mathrm{sk} \mod n
\end{aligned}
$$

What is the problem with solving this? How can we fix it?

Partially Equal Randomness

Subtracting equation 3 from 1 and 2 gives:

$$b_1 - b_3 \equiv (s_1^{-1} h_1 - s_3^{-1} h_3) + (s_1^{-1} r_1 - s_3^{-1} r_3) \cdot \mathrm{sk} \mod n$$
$$b_2 - b_3 \equiv (s_2^{-1} h_2 - s_3^{-1} h_3) + (s_2^{-1} r_2 - s_3^{-1} r_3) \cdot \mathrm{sk} \mod n$$

The unknown $2^{128}a$ cancels, and $b_i - b_3$ is still short, of absolute value less than 2^{128}. Every other coefficient is of size n, so the same lattice construction still works, with one dimension less: ℓ signatures give $\ell - 1$ equations.

Partially Equal Randomness

$$\begin{bmatrix} j_1 & j_2 & \mathsf{sk} & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ t_1 - t_3 & t_2 - t_3 & B/n & 0 \\ u_1 - u_3 & u_2 - u_3 & 0 & B \end{bmatrix} = \begin{bmatrix} b_1 - b_3 & b_2 - b_3 & \mathsf{sk} \cdot B/n & B \end{bmatrix}$$

When this basis is LLL-reduced, its first vector is, with high probability, the shortest vector in the lattice, and that vector again contains the secret key sk.

Contents

Setting Up Our Parameters (secp256k1)

```
p = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F
E = EllipticCurve(GF(p), [0, 7])
G = E([0
    ↪x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798
    ↪, 0
    ↪x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8
    ↪])
n = G.order()
nl = int(n).bit_length()
## Create private key
sk = randrange(1, n-1)
## Create public key
pk = sk*G
## Function to reduce mod n
N = Zmod(n)
```

Signing Messages

```
# Number of messages we capture
nsigs = 3
messages = [f"message {i}".encode() for i in range(nsigs)]

## Length of randomness used
T = 2^128
K = [randrange(1, T) for _ in range(nsigs)]
print(K)
H = [int.from_bytes(sha256(msg).digest()[:nl//8], "big") for msg in
    ↪messages]

Points = [int(K[i])*G for i in range(nsigs)]
X = [P[0] for P in Points]
R = [N(x) for x in X]
S = [(H[i] + sk*R[i])/N(K[i]) for i in range(nsigs)]
```

Recovering the Key

- The code is omitted here, since this is very close to one of the CryptoHack problems on the weekly problem sheet.
- The attack itself relies on an existing implementation of LLL.
- We recommend SageMath, Python library `fpylll`, or C library `fplll`.
- Construct the lattice basis described earlier, reduce it with LLL, and finally recover the secret key by simple arithmetic.

How Fast Is It?

For small lattices, where the nonces are very short or share large parts of their bits, the attack is fast: a matter of seconds on a laptop.

```
Curve: Elliptic Curve defined by y^2 = x^3 + 7 over Finite Field of size 115792089237316195423570985008687907853269984665640564039457584007908834671663
p: 115792089237316195423570985008687907853269984665640564039457584007908834671663 0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
n: 115792089237316195423570985008687907852837564279074904382605163141518161494337 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
d: 93014717667518195997201708971372419465881181548562368794432875239154275681886 0xcda476ecc4a3cf5012534f99e6b85f1852442c71fd1741ec308db39e591d865e
G: (0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798, 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8)
Q: (0x130e99394b6358f01bf55cfb7daf10faaa1de4552b8b719db89f4c1aa56de88, 0x7ee04c1622da77d9868574eee8b812c2d7be6a519eea0a6a28f3719201277618)
[338312366969595278585886726037755088320, 1181316190540772705004638401601311839O3, 13765855804792531697770441694878307329 7]
------ Attack ------
--------------------


Found d: 93014717667518195997201708971372419465881181548562368794432875239154275681886
sage lattice_attack_demo.sage  0.85s user 0.15s system 102% cpu 0.980 total
```

Biased Nonce Sense

Biased Nonce Sense: Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies

Joachim Breitner[1 [0000−0003−3753−6821]] and Nadia Heninger[2]

Figure: https://eprint.iacr.org/2019/023

Only Half the Nonce Is Enough

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]

Figure: https://eprint.iacr.org/2023/841

LadderLeak: Breaking ECDSA With Less Than One Bit Of Nonce Leakage

Diego F. Aranha
DIGIT, Aarhus University
Denmark
dfaranha@eng.au.dk

Felipe Rodrigues Novaes
University of Campinas
Brazil
ra135663@students.ic.unicamp.br

Akira Takahashi
DIGIT, Aarhus University
Denmark
takahashi@cs.au.dk

Mehdi Tibouchi
NTT Corporation
Japan
mehdi.tibouchi.br@hco.ntt.co.jp

Yuval Yarom
University of Adelaide and Data61
Australia
yval@cs.adelaide.edu.au

Figure: https://eprint.iacr.org/2020/615

The “Amaclin” Case

- “Amaclin” is the screen name of a user on several forums.
- They have tricked a number of people into using weak nonces, and have most likely stolen a considerable amount of money.
- In one case, users were persuaded to derive the nonce from part of their own secret key together with fresh randomness, which enabled an attack very similar to the ones in this lecture.

Contents

How to Not Lose Your Key

Do not trust the PRNG

- **Deterministic nonces (RFC 6979).** Derive $k = \mathrm{H}(\mathrm{sk}, m)$ with HMAC instead of sampling it, so a broken PRNG cannot repeat or shorten the nonce.
- **EdDSA (RFC 8032).** Ed25519 is deterministic by design, and fixes the curve, the encoding, and the hash as well. Increasingly used today.
- **Hedged signing** is the compromise: $k = \mathrm{H}(\mathrm{sk}, m, \text{fresh randomness})$, which survives both a bad PRNG and potential side-channel attacks.

Contents

What to Remember

- The nonce must be fresh, uniform, and secret: two signatures sharing one hand over the key.
- Partial knowledge is enough as well, from short nonces, a shared prefix, or a side channel.
- That is a hidden number problem, and LLL turns it into key recovery in seconds.
- Derive the nonce (RFC 6979, Ed25519), hedge it, and keep the scalar multiplication constant-time

Questions?