Summer School on real-world crypto and privacy 2023

Practical Side-Channel Attacks on Real-World ECDSA Implementations

Ján Jančár j08ny@mail.muni.cz
Łukasz Chmielewski chmiel@fi.muni.cz
Centre for Research on Cryptography and Security, Masaryk University

CRoCS Centre for Research on Cryptography and Security

About us

- CRoCS, Masaryk University
- Ján Jančár
  - PhD Candidate @ CRoCS
  - https://neuromancer.sk/
- Łukasz Chmielewski
  - Assistant Professor @ CRoCS
  - Interested in practical aspects of SCA and FI

Thank you to Milan Šorf for help!

Main Goals

- Give you an impression of the attack surface on the Elliptic Curve Digital Signature (ECDSA) algorithm with respect to Side-Channel Analysis and Fault Injection (FI)

- Give you practical experience on nonce-related attacks on ECDSA.

- Describe the Minerva vulnerability and show how to exploit
  - Exercises!

- **Disclaimer:** there will be a lot of technical details so it OK not to grasp everything

  - This is NOT a tutorial on ECC, ChipWhisperer, generic SCA…

Outline

- A (very) brief introduction to:
  - Side-Channel Analysis (SCA) and Fault Injection (FI)
  - Elliptic Curve Cryptography (ECC)
- ECDSA & (Selected) Attacks on ECDSA

- Practical Exercises
  - Did everyone install the tooling / VM?

- Countermeasures & Conclusions

(VERY BRIEF) INTRO TO SIDE-CHANNEL ANALYSIS AND FAULT INJECTION

Why Is Hardware Security Important?

Card / Money Theft

Identity Theft

- **Premium Content**

Phone / Money Theft

Impersonation

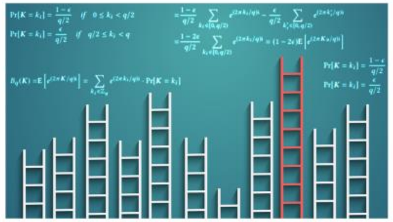
Recent Practical Attacks

November 13, 2019

May 28, 2020

LadderLeak: Side-channel security flaws exploited to break ECDSA cryptography

Existing attacks refined to attack elliptic curve algorithm



SCA Titan: January 7, 2021

October 3, 2019

Researchers Discover ECDSA Key Recovery Method

December 12, 2019

PLUNDERVAULT —

Intel's SGX coughs up crypto keys when scientists tweak CPU voltage

Install fixes when they become available. Until then, don't sweat it.

DAN GOODIN - 12/10/2019, 11:41 PM

Introduction: Side Channel Analysis

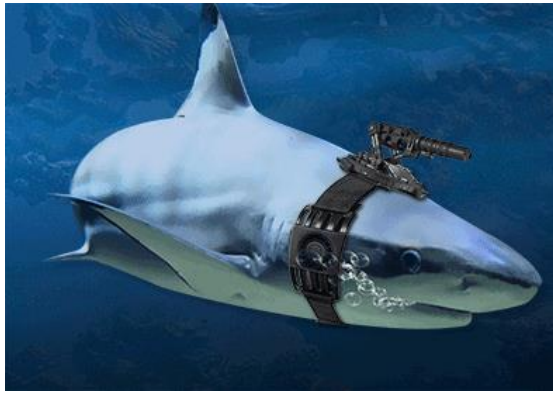
Side Channels

- Time
- Power
- Electro Magnetic Emanations
- Light
- Sound
- Temperature
- …

Passive vs Active Side Channels

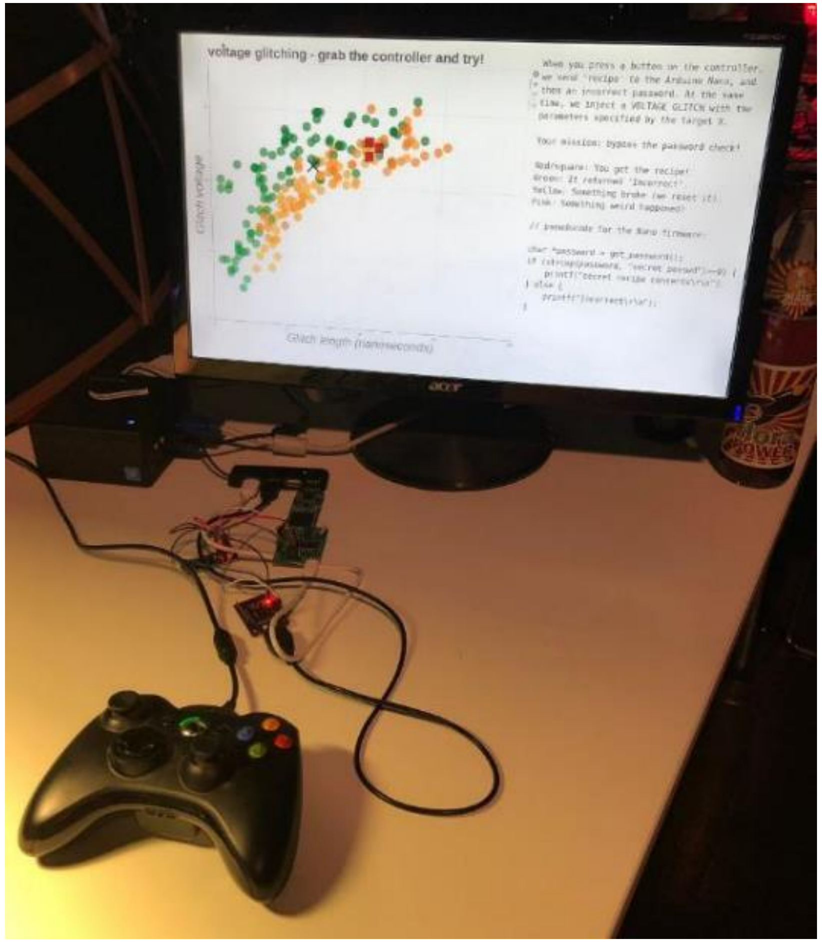
Passive: analyze device behavior

Active: change device behavior



https://escooptics.com/blogs/news/world-space-week-02-lasers

Some Practical Side-Channel Setups



“Commercial” Example: the “unlooper” device

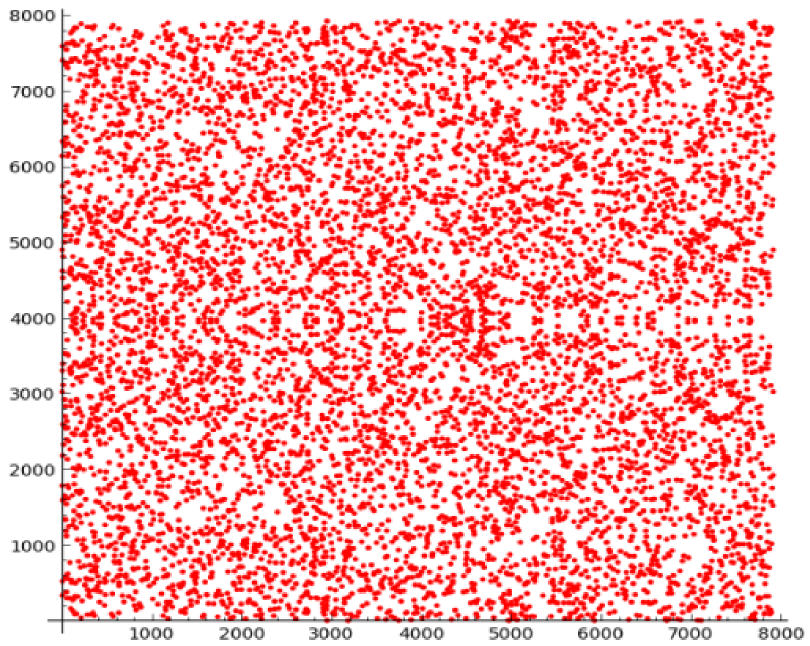
```c
void entry() {
    void* start  = 0x80000000;
    void* length = 0x00400000;

    serial_puts("Start Secure Boot...\n");

    loadOSFromHardDrive(start);

    if (! authenticateOS(start,length) )
        do {} while(1);

    serial_puts("Run OS\n");

    boot_next_stage(start);
    //starts executing at the address start
}
```

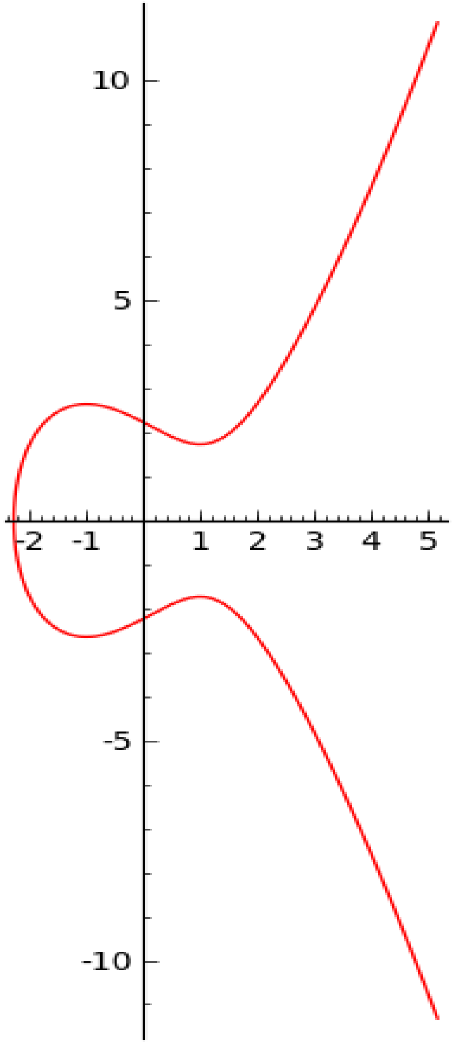
(VERY BRIEF) INTRO TO ELLIPTIC CURVE CRYPTOGRAPHY

Elliptic curves

- Example
  - $y^2 = x^3 - 3x^2 + 5$
- How would it look: *mod* 7919?

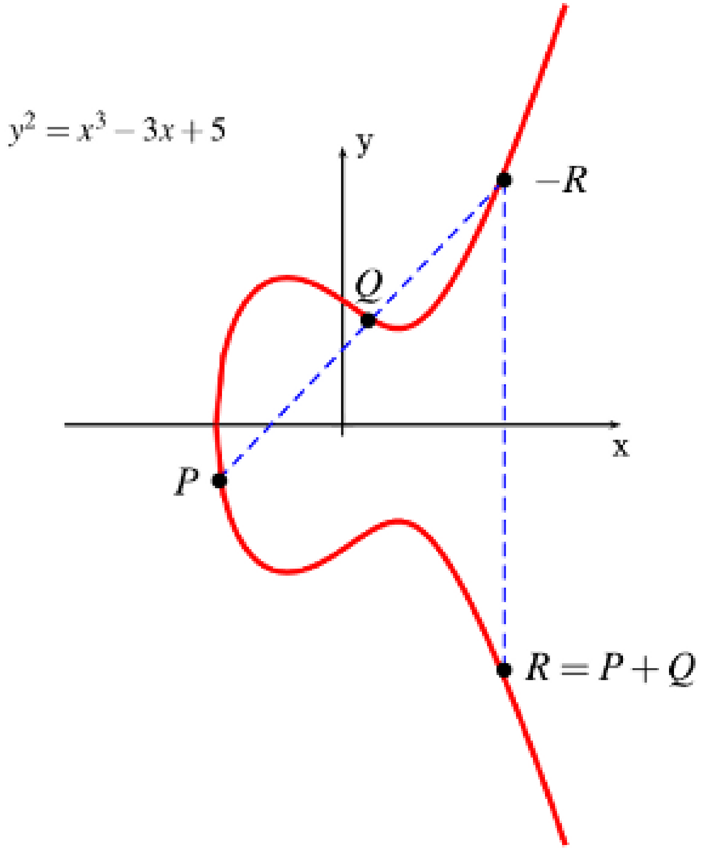
(x, y)



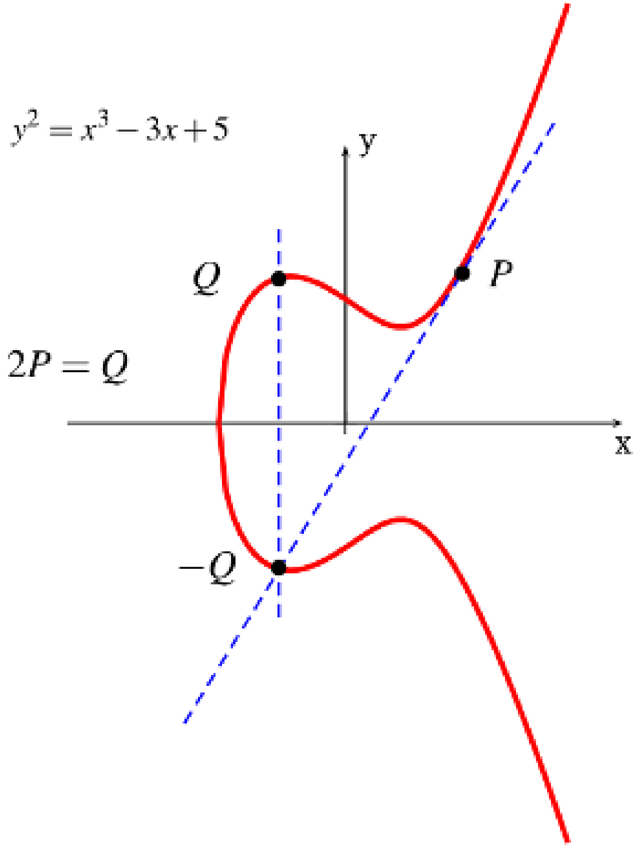


Elliptic curve arithmetic - an abelian group

Addition



Doubling



Scalar multiplication by *k*

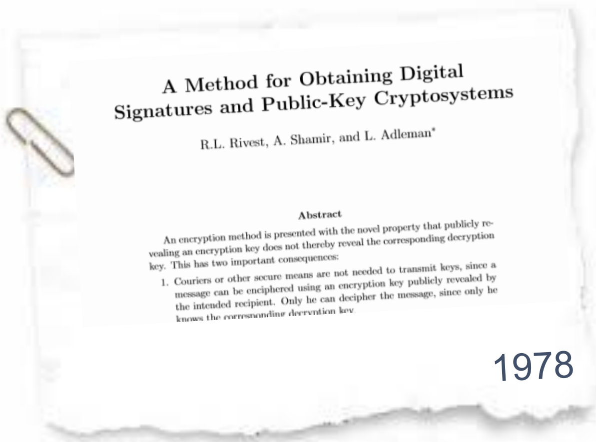
- Group = points on the curve + special point ∞
- Add a point to itself k times: Q = [k]P
- Analogous to modular exponentiation
  - square-and-multiply = double-and-add
- Many (equivalent) algorithms
  - LTR, RTL, Window, Comb, Ladders, …

```
x=∞
for j=|k|-1 to 0 {
   x=DBL(x)
  if k_j=1
   x=ADD(x,P)
return x_0
```

```
(0xb1011) = 11
x=∞
j=3
  x=DBL(∞)
  x=ADD(∞,P)
j=2
  x=DBL(P)
j=1
  x=DBL(2P)
  x=ADD(4P,P)
j=0
  x=DBL(5P)
  x=ADD(10P,P)=11P
```

RSA ≈ ECC

- RSA is based on modular exponentiation
- exponentiation ≈ scalar multiplication
  - $m = c^d \bmod n \approx Q = [k]P$
- multiplication ≈ points addition
- squaring ≈ point doubling


A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

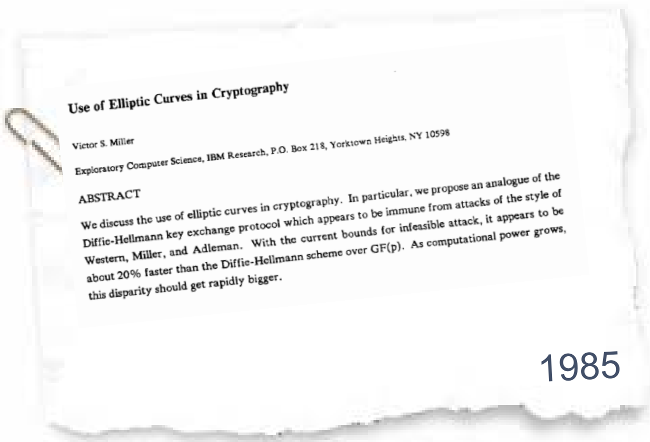
R.L. Rivest, A. Shamir, and L. Adleman*

Abstract

An encryption method is presented with the novel property that publicly revealing an encryption key does not thereby reveal the corresponding decryption key. This has two important consequences:

1. Couriers or other secure means are not needed to transmit keys, since a message can be enciphered using an encryption key publicly revealed by the intended recipient. Only he can decipher the message, since only he knows the corresponding decryption key.

1978


Use of Elliptic Curves in Cryptography

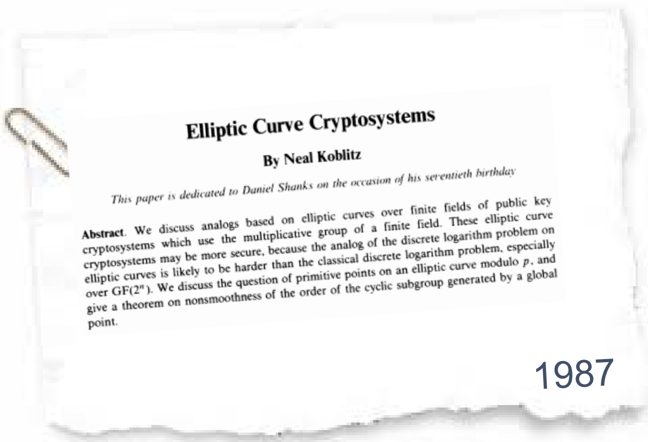
Victor S. Miller

Exploratory Computer Science, IBM Research, P.O. Box 218, Yorktown Heights, NY 10598

ABSTRACT

We discuss the use of elliptic curves in cryptography. In particular, we propose an analogue of the Diffie-Hellmann key exchange protocol which appears to be immune from attacks of the style of Western, Miller, and Adleman. With the current bounds for infeasible attack, it appears to be about 20% faster than the Diffie-Hellmann scheme over GF(p). As computational power grows, this disparity should get rapidly bigger.

1985


Elliptic Curve Cryptosystems

By Neal Koblitz

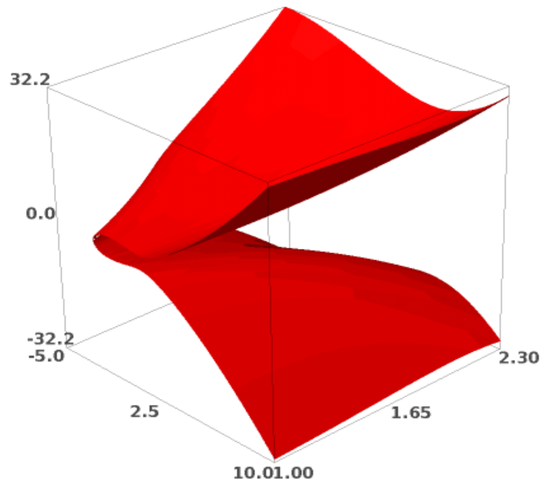
This paper is dedicated to Daniel Shanks on the occasion of his seventieth birthday

Abstract. We discuss analogs based on elliptic curves over finite fields of public key cryptosystems which use the multiplicative group of a finite field. These elliptic curve cryptosystems may be more secure, because the analog of the discrete logarithm problem on elliptic curves is likely to be harder than the classical discrete logarithm problem, especially over $GF(2^n)$. We discuss the question of primitive points on an elliptic curve modulo p, and give a theorem on nonsmoothness of the order of the cyclic subgroup generated by a global point.

1987

Implementation Considerations

- Some curves are more common than others:
  - NIST curves (p256), Curve25519, or secp256k1
- Many security properties
  - Curves classification wrt. security: http://safecurves.cr.yp.to/
- Are there powerful fault attacks against ECDSA?
  - Invalid point attacks
  - Attacks against deterministic ECDSA
- The classical (x,y) representation is rarely used
  - For transport points compression is often used (note that x defines y)
  - For efficiency reasons it is better to represent a point in more dimensions (e.g., (x,y,z))



What are the applications of ECC?

- Cryptocurrencies

- Signatures: ECDSA and deterministic variants
- Key Exchange: ECDH
- Encryption: ECIES

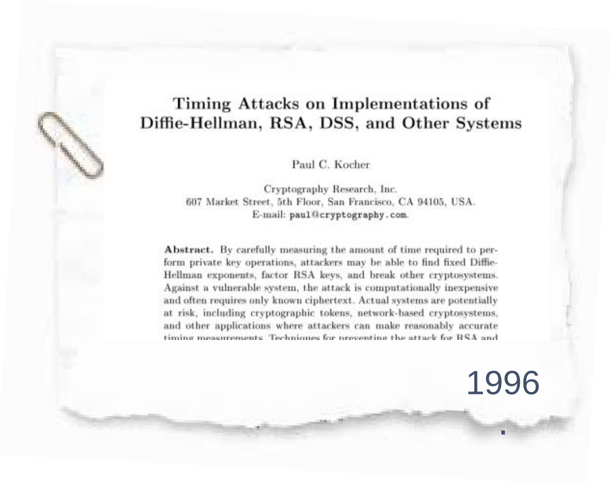
- In this tutorial we concentrate on ECDSA

"CLASSICAL" SCA ON SCALAR MULTIPLICATION

Simple Power Analysis (SPA) on ECC (scalarmult)

```
ScalarMult(P){

 A = ∞
 for ( i = n-1;  i0; i)
           A = DOUBLE(A)
           if (k_i = = 1)
                     A = ADD(A,P)
           end if
 end for
 Return A = [k]P
}
```

D
A



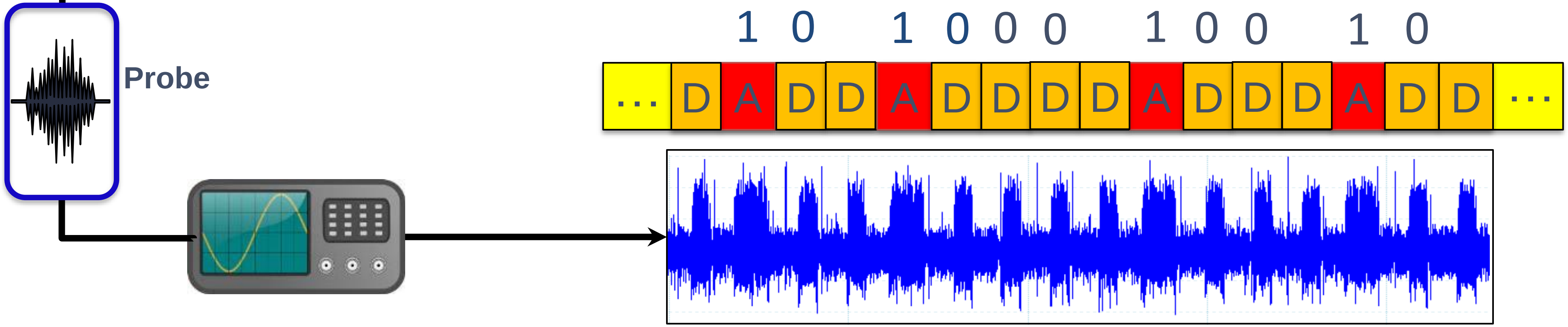
Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems

Paul C. Kocher

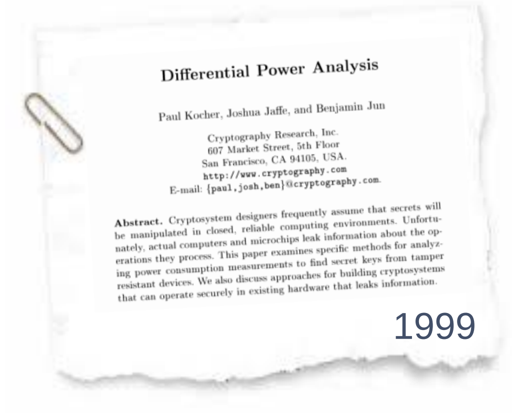
Cryptography Research, Inc.
607 Market Street, 5th Floor, San Francisco, CA 94105, USA.
E-mail: paul@cryptography.com.

Abstract. By carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems. Against a vulnerable system, the attack is computationally inexpensive and often requires only known ciphertext. Actual systems are potentially at risk, including cryptographic tokens, network-based cryptosystems, and other applications where attackers can make reasonably accurate timing measurements. Techniques for preventing the attack for RSA and

1996



Correlation Power Analysis on RSA/ECC

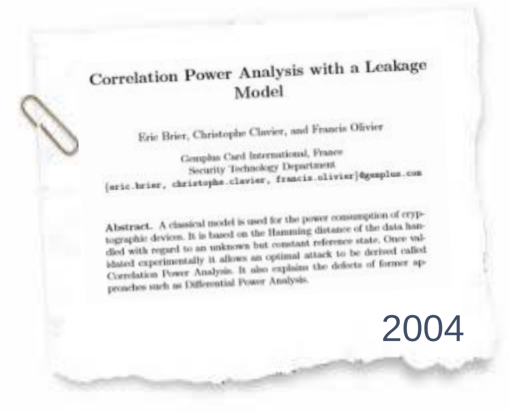

Differential Power Analysis

Paul Kocher, Joshua Jaffe, and Benjamin Jun

Cryptography Research, Inc.
607 Market Street, 5th Floor
San Francisco, CA 94105, USA.
http://www.cryptography.com
E-mail: {paul,josh,ben}@cryptography.com

Abstract. Cryptosystem designers frequently assume that secrets will be manipulated in closed, reliable computing environments. Unfortunately, actual computers and microchips leak information about the operations they process. This paper examines specific methods for analyzing power consumption measurements to find secret keys from tamper resistant devices. We also discuss approaches for building cryptosystems that can operate securely in existing hardware that leaks information.

1999

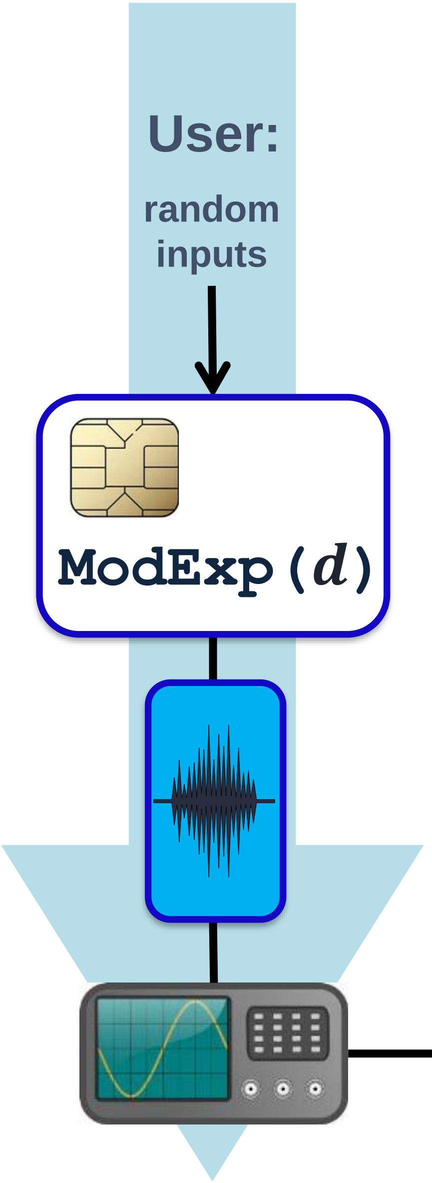

Correlation Power Analysis with a Leakage Model

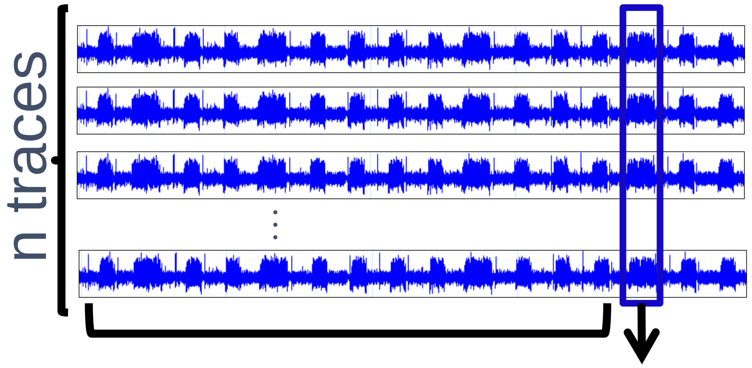
Eric Brier, Christophe Clavier, and Francis Olivier

Gemplus Card International, France
Security Technology Department
{eric.brier, christophe.clavier, francis.olivier}@gemplus.com

Abstract. A classical model is used for the power consumption of cryptographic devices. It is based on the Hamming distance of the data handled with regard to an unknown but constant reference state. Once validated experimentally it allows an optimal attack to be derived called Correlation Power Analysis. It also explains the defects of former approaches such as Differential Power Analysis.

2004





f_i = **Selection Function**(random inputs, $\hat{d}$, target state)

- HW of a state in a register
- HD between current and previous register state
- ID model: value of a register
- Most significant bit:

$$f_i = \begin{cases} 0 & if\ HW \leq 16 \\ 1 & if\ HW > 16 \end{cases}$$

$$f_i = HW(reg\_state)$$

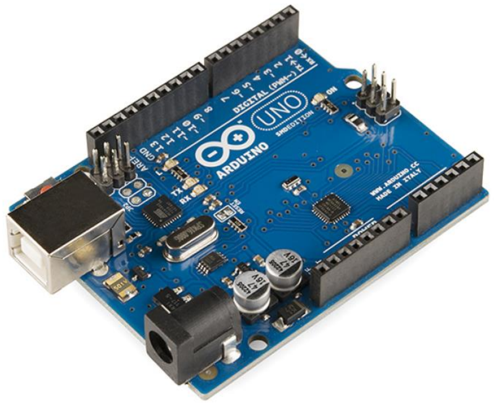
DPA = Difference of Means

CPA = Pearson correlation

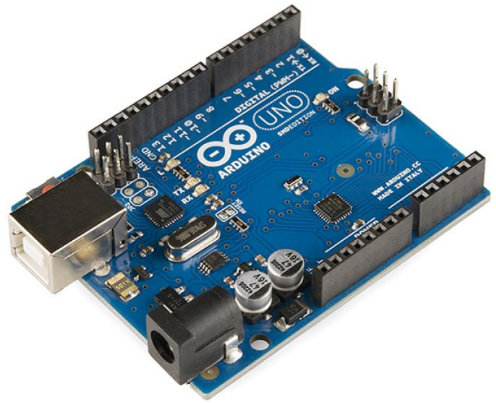
Profiled Attacks

- Problems with the above approaches:
  - can we attack the key directly?
  - we often do not get many traces with the same secret
  - can we use an unprotected device of the same model?
- (Possible) Solution:
  - We profile, i.e. template the unprotected device
  - We use the profile to break the protected device
- Procedure:
  1. Choose a model that describes the power consumption
  2. Profile the unprotected device to create the template (Template Building)
  3. Use the template to break the protected device (Template Matching)
- The same steps are always performed but the model can be different.
  - So often we will not learn the secret but the hamming weight of the secret.
- Neural Networks can be used instead of Template Attacks

Controlled unprotected device



Attacked protected device



ECDSA

Introduction

- A variant of the Digital Signature Algorithm (DSA)
- Signature Algorithm based on ECC used, among others, for:
  - Cryptocurrencies,
  - Secure boot
- Problems:
  - Backdoors (?), technical issues (hard to implement), quantum computers
- Libraries:
  - Bouncy Castle, mbed TLS, Microsoft CryptoAPI, OpenSSL, wolfCrypt, and many more…
- We omit analysis of key generation:
  - Alice creates a key pair:
    - a private key: random integer x from [1,q-1], where q is the curve order and
    - a public key curve point Q=[d]P, where P is a group generator (constant).

ECDSA Algorithm

- Signature Generation (secret: x; input: m; output: r, s)
  1. A random number k, $1 \leq k \leq q-1$ is chosen
  2. Compute $[k]P = (x_1,y_1)$ is computed.
  3. Next, $r = x_1 \bmod q$ is computed.
  4. We then compute $k^{-1} \bmod q$
  5. $s = k^{-1}(H(m) + \underline{d}*r) \bmod n$

- Signature Verification (input: m, r, s; output: True/False)
  1. Verify r and s are in [1, q-1] for the signature to be valid.
  2. Compute $u_1 = H(m)*s^{-1} \bmod q$ and $u_2 = r*s^{-1} \bmod q$.
  3. Compute $(x_1,y_1) = [u_1]G + [u_2]Q$.
  4. The signature is valid if $r = x_1 \bmod q$, invalid otherwise.

But what to attack?

Functional: Randomness needs to be good!!!

Lattice attacks on the nonces. More about that later.

ECDSA Algorithm – Passive SCA

- Signature Generation (secret: x; input: m; output: r, s)
  1. A random number k, $1 \le k \le q\text{-}1$ is chosen
  2. Compute $[k]P = (x_1,y_1)$ is computed.
  3. Next, $r = x_1 \bmod q$ is computed.
  4. We then compute $k^{-1} \bmod q$
  5. $s = k^{-1}(H(m) + \underline{d}*r) \bmod n$

- Signature Verification (input: m, r, s; output: True/False)
  1. Verify r and s are in [1, q-1] for the signature to be valid.
  2. Compute $u_1 = H(m)*s^{-1} \bmod q$ and $u_2 = r*s^{-1} \bmod q$.
  3. Compute $(x_1,y_1) = [u_1]G + [u_2]Q$.
  4. The signature is valid if $r = x_1 \bmod q$, invalid otherwise.

+ Lattice attacks on the nonces.

ECDSA Algorithm – Fault Injection

- Signature Generation (secret: x; input: m; output: r, s)
  1. A random number k, 1 ≤ k ≤ q-1 is chosen
  2. Compute [k]P = (x_1,y_1) is computed.
  3. Next, r = x_1 mod q is computed.
  4. We then compu
  5. s = k^{-1}(H(m) +

- Signature Verifica
  1. Verify r and s a
  2. Compute u_1 = H
  3. Compute $(x_1,y_1$
  4. The signature is

To avoid randomness issues, can ECDSA be done without randomness?
Yes, but it has its own risks…

+ Lattice attacks on the nonces.

ECDSA AND NONCES
MINERVA
EXCERCISES

COUNTERMEASURES/MITIGATIONS

SUMMARY

Generic protection techniques

1. Do not leak
   – Constant-time crypto, bitslicing…
2. Shielding - preventing leakage outside
   – Acoustic shielding, noisy environment
3. Creating additional “noise”
   – Parallel software load, noisy power consumption circuits
4. Compensating for leakage
   – Perform inverse computation/storage
5. Prevent leaking exploitability
   – Ciphertext and key blinding, key regeneration, masking of the operations

Specific protection techniques

$$M = [s]P = [s](X, Y) = [s](x, y, 1)$$

1. $M = [s](x.z, y.z, z)$	coordinate blinding
2. $s_r = s + r * \lvert E \rvert$	scalar blinding
3. $M_r = [s_r](x.z, y.z, z)$	blinded scalar mult.
4.	no unblinding

The sequence of operations (D, A) is related to the scalar bits.

However:

- If s is random: the sequence of scalar bits changes for every scalar multiplication execution
- If P is random: Intermediate data is random (masked) -> hardly predicted!

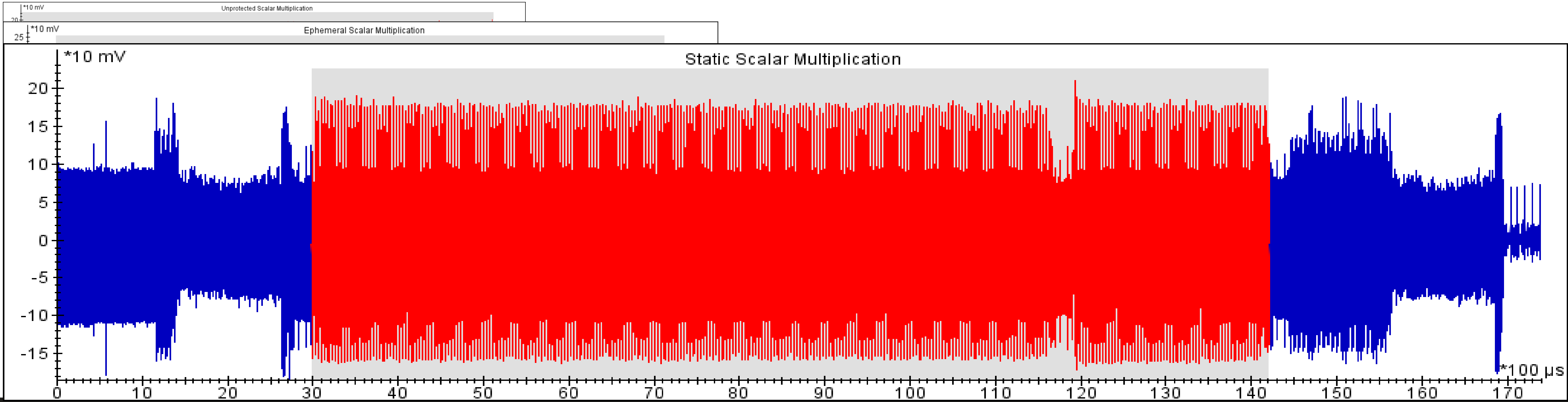
DPA is based on the prediction of intermediate data. We hope that: any side-channel attack requiring multiple traces is repelled by message and exponent blinding.

Point blinding is also possible to prevent zero-point attacks but not presented above.

There are many more similar countermeasures, e.g. scalar splitting.

Scalar Multiplication Countermeasures

- No public (open source) protected library for ECDSA
  - There are partially protected libaries
- For ECDH situation is a bit better, for example:
  - [BCHSP2023] L. Batina, Ł. Chmielewski, B. Haase, N. Samwel, and P. Schwabe: SoK: SCA-secure ECC in software – mission impossibile? CHES 2023



Differences between ECDH and ECDSA scalar mult.

- Can secure ECDH Implementation be used for ECDSA scalar multiplication? What could be the problem?

```c
int crypto_scalarmult_curve25519(uint8_t *r, const uint8_t *s,
                                 const uint8_t *p) {
  ST_curve25519ladderstepWorkingState state;
  uint8_t i;
  volatile uint8_t retval = -1;
  // Fault injection detection counter ### alg. step 1 ###
  volatile uint32_t fid_counter = 0;

  // Initialize return value with random bits ### alg. step 2 ###
  randombytes(r, 32);

  // Prepare the scalar within the working state buffer.
  for (i = 0; i < 32; i++) {
    state.s.as_uint8_t[i] = s[i];
  }

  // Copy the affine x-coordinate of the base point to the state.
  fe25519_unpack(&state.x0, p);

  fe25519_setone(&state.xq);
  fe25519_setzero(&state.zq);
  fe25519_cpy(&state.xp, &state.x0);
  fe25519_setone(&state.zp);

  // Clamp scalar ### alg. step 3 ###
  state.s.as_uint8_t[31] &= 127;
  state.s.as_uint8_t[31] |= 64;

  // ### alg. step 4 ###
  shiftRightOne(&state.s);
  shiftRightOne(&state.s);
```

Additional ECDSA countermeasures

1. Blind modular multiplication x * r (in $s = k^{-1} * (H(m) + x*r) \bmod q$)
   - Multiplicative blinding
2. Protect Inversion
   - Use the Bernstein-Yang inversion constant-time inversion algorithm
   - Multiplicative blinding
3. Protect most significant bits:
   - Scalar randomization or deterministic way:
     - Conditionally adding the group order q or 2q to the scalar k until the resulting scalar has a fixed length.
4. Point blinding
5. Still blind the nonce!
6. Add generic protections against FI:
   - Redundancy, control flow protections …

And much more…

CONCLUSIONS

Conclusions

- ECDSA is vulnerable to attacks on nonces, in particular lattice attacks.
  - Minerva was discovered on certified products.
- Protecting ECDSA against SCA and FI is hard
- Good randomness is crucial! Protect the nonces!

Questions 

MUNI
FI

CROCS
Centre for Research on Cryptography and Security

June 9, 2023

Practical part: ECDSA and nonces

Jan Jancar
Łukasz Chmielewski

ECDSA nonce k

- Needs to be unpredictable & unknown to an attacker
- What happens if it is not?
  - Nonce (full) leak
  - Nonce reuse
  - Nonce bias
  - Nonce (partial) leak

$$k \xleftarrow{\$} \mathbb{Z}_n^*$$

ECDSA nonce k

- Needs to be unpredictable & unknown to an attacker
- ? What happens if it is not?
  - Nonce (full) leak
  - Nonce reuse
  - Nonce bias
  - Nonce (partial) leak
- ! Answer: Private key recovery
  - Number of signatures required $2 \ldots 2^{32}$

$$k \xleftarrow{\$} \mathbb{Z}_n^*$$

Nonce leak

- (r, s) is the signature, m is the message
- Knowing k, we can compute x
- Why would this happen?

1 signature

$$r \equiv ([k]G)_x \qquad \bmod n$$
$$s \equiv k^{-1}(H(m) + rx) \qquad \bmod n$$

$$x \equiv (ks - H(m))r^{-1}$$

Nonce leak

- (r, s) is the signature, m is the message
- Knowing k, we can compute x
- Why would this happen?
  - Clueless developer?

1 signature

$$r \equiv ([k]G)_x \qquad \mod n$$
$$s \equiv k^{-1}(H(m) + rx) \qquad \mod n$$

$$x \equiv (ks - H(m))r^{-1}$$

Nonce reuse

- Subtract $s_1 - s_2$, obtain k
- From there get x as before
- Why would this happen?

2 signatures

$$r \equiv ([k]G)_x$$

$$s_1 \equiv k^{-1}(H(m_1) + rx)$$

$$s_2 \equiv k^{-1}(H(m_2) + rx)$$

$$s_1 - s_2 \equiv k^{-1}(H(m_1) - H(m_2))$$

$$k \equiv \frac{H(m_1) - H(m_2)}{s_1 - s_2}$$

Nonce reuse

- Subtract $s_1 - s_2$, obtain k
- From there get x as before
- Why would this happen?
  - Clueless developer?
  - Static nonce? (PlayStation 3) [1]
  - Bad RNG?

2 signatures

$$r \equiv ([k]G)_x$$
$$s_1 \equiv k^{-1}(H(m_1) + rx)$$
$$s_2 \equiv k^{-1}(H(m_2) + rx)$$

$$s_1 - s_2 \equiv k^{-1}(H(m_1) - H(m_2))$$
$$k \equiv \frac{H(m_1) - H(m_2)}{s_1 - s_2}$$

Nonce bias $\approx$ partial leak

- Some bias is like an information leak
- 2 attacks
  - Lattice reduction (HNP)
  - "Bleichenbacher"
- Why would this happen?

$2 \ldots 2^{32}$ signatures

$$k < n/2^l$$

$$k \equiv 5 \mod 8$$

Nonce bias $\approx$ partial leak

- Some bias is like an information leak
- 2 attacks
  - Lattice reduction (HNP)
  - “Bleichenbacher”
- Why would this happen?
  - Bad RNG? [2]
  - Bad constant-timeness fix?
  - Side-channel leak?

$2 \ldots 2^{32}$ signatures

$$k < n/2^l$$

$$k \equiv 5 \mod 8$$

Nonce bias $\approx$ partial leak

Lattice reduction (HNP)

- Hidden Number Problem
- ≥ 2 bits of info
- Small number of signatures
- Transforms into CVP or SVP
- [3,4,5]

Bleichenbacher

- Bias amplification + search
- Even 1 bit of info
- Large number of signatures
- [6]


Minerva
Leaky loops
Leaky loops
Ingredients: Athena IDProtect, Libgcrypt, SunEC, wolfSSL, MatrixSSL. May contain traces of bits
Ján Jančár
Vladimír Sedláček
Petr Švenda
Marek Sýs
9 780201 379624
CROCS
Centre for Research on Cryptography and Security

Minerva

Discovery: ECDSA testing

- ASN.1 parsing ✅

Minerva

Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅

Minerva

Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~

Minerva

Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅

Minerva

Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅

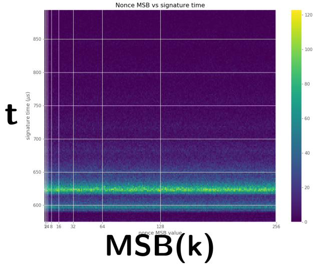
Let’s test timing as well!

Minerva

Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅

Let’s test timing as well!

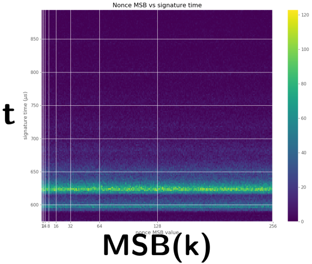


Minerva

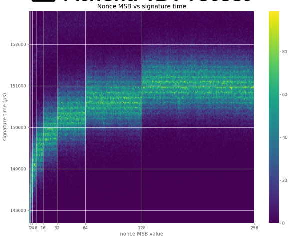
Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅

Let's test timing as well!



Athena IDProtect

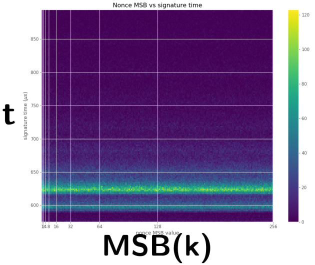


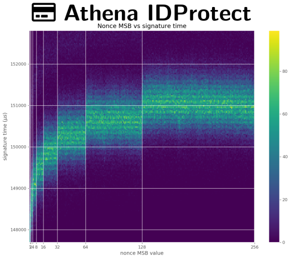
Minerva

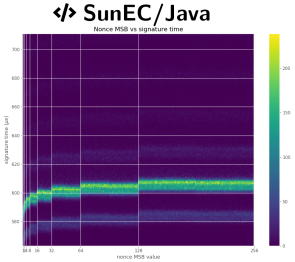
Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅

Let's test timing as well!





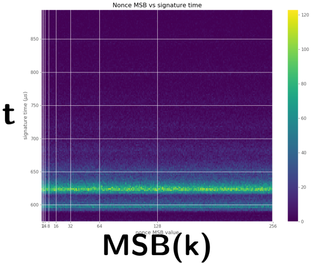


Minerva

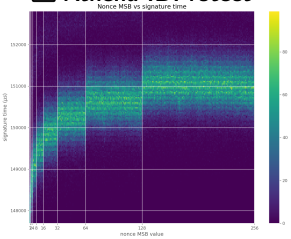
Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅

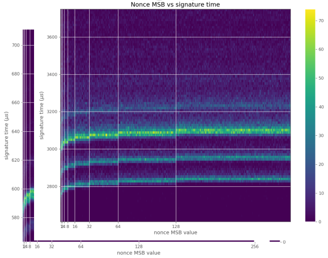
Let's test timing as well!



Athena IDProtect



libgcrypt

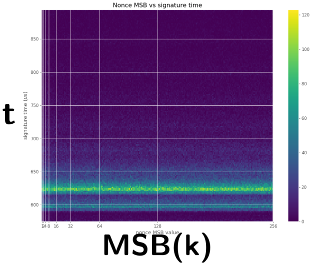


Minerva

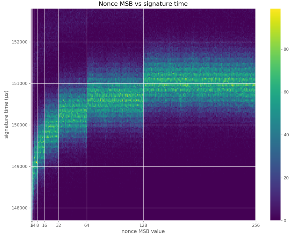
Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅

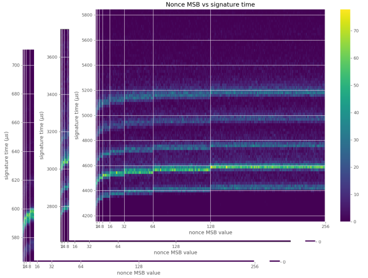
Let's test timing as well!



Athena IDProtect



MatrixSSL

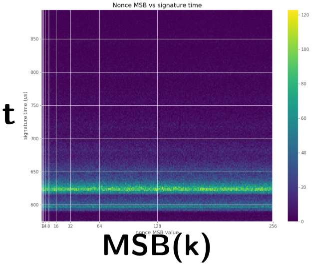


Minerva

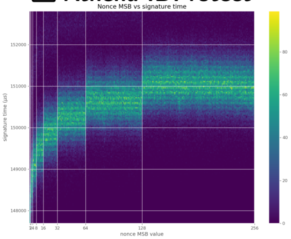
Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅

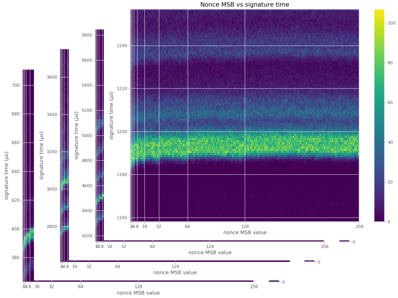
Let's test timing as well!



Athena IDProtect



WolfSSL

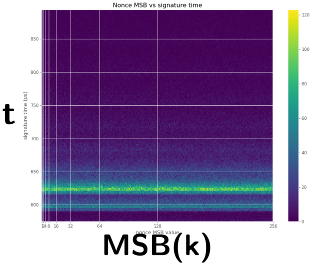


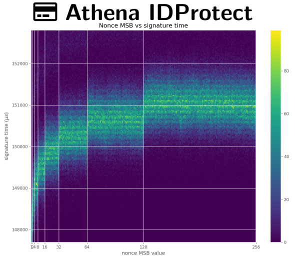
Minerva

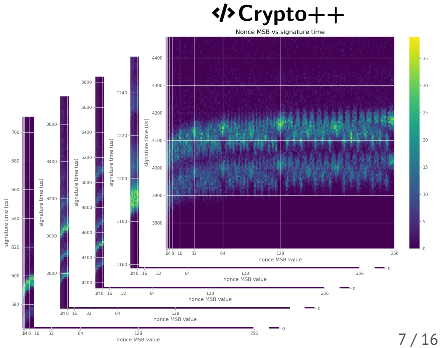
Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅

Let's test timing as well!



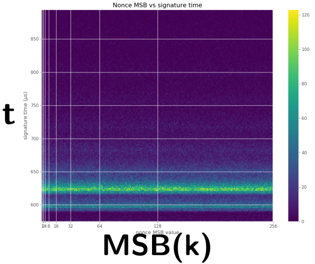




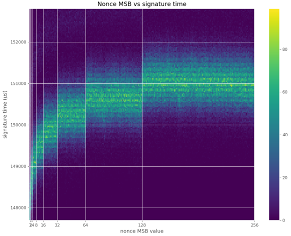
Minerva

Discovery: ECDSA testing

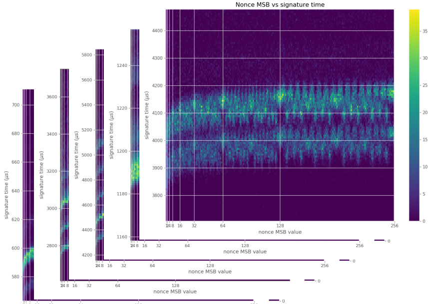
- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅
- Timing ❌



Athena IDProtect



Crypto++



Minerva

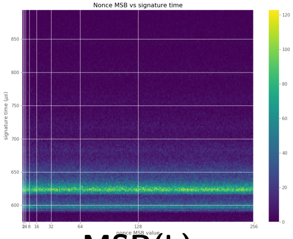
Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅
- Timing ❌

Minerva

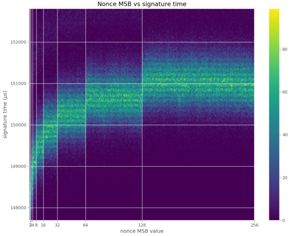
1
5

t

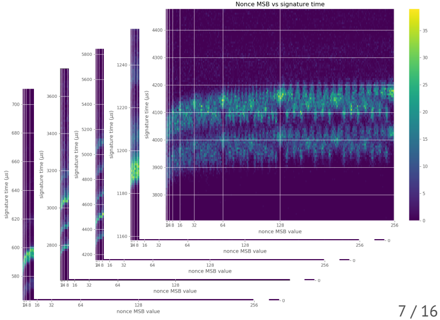


MSB(k)

Athena IDProtect



</> Crypto++



Minerva

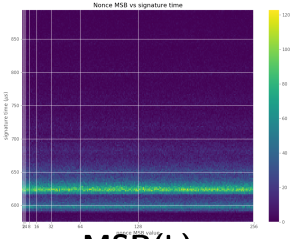
Discovery: ECDSA testing

- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅
- Timing ❌

Minerva 1 5

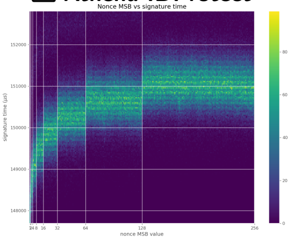
TPM-FAIL 2

t

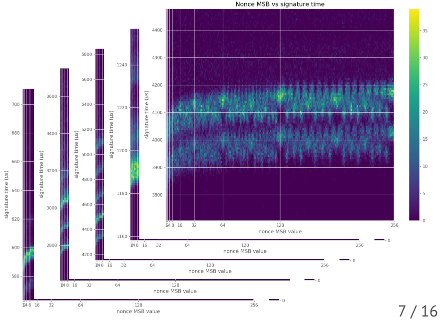


MSB(k)

Athena IDProtect



Crypto++



Minerva

Discovery: ECDSA testing

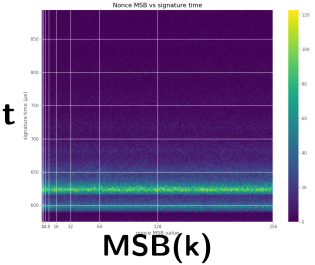
- ASN.1 parsing ✅
- Signature malleability ✅
- Test-vectors ~
- Nonce randomness ✅
- Timing ❌

Minerva 1 5

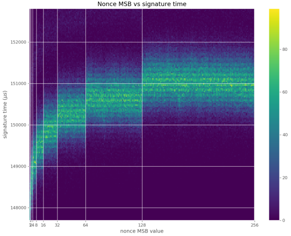
TPM-FAIL 2

Déjà Vu 1

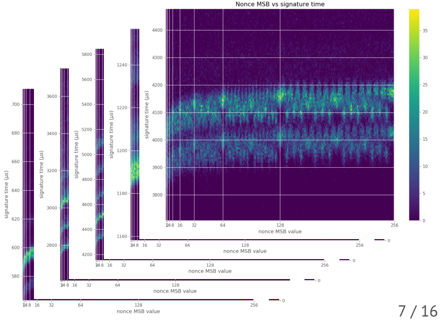
...



Athena IDProtect



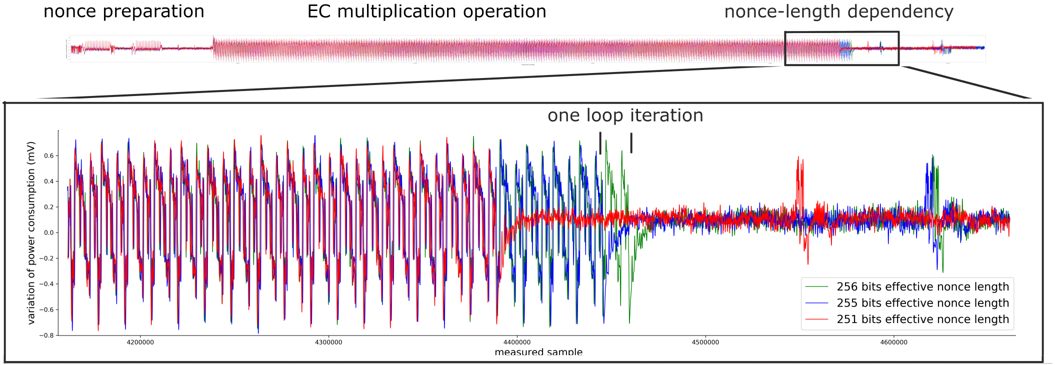
</> **Crypto++**



Discovery

Leak

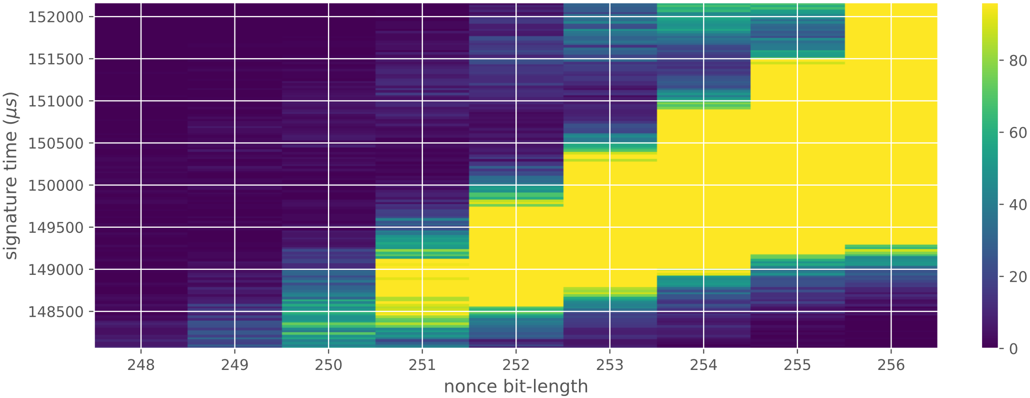
[k]G



Discovery

Leak

[k]G



Exploitation

Hidden Number Problem

- Average 1 LZB per signature
- There is noise

Exploitation

Hidden Number Problem

- Average 1 LZB per signature
- There is noise

Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes

(Extended Abstract) [7]

DAN BONEH[1]
Princeton University

RAMARATHNAM VENKATESAN[2]
Bellcore

Exploitation

Hidden Number Problem

- Average 1 LZB per signature
- There is noise

Hidden Number Problem (HNP) [1]

Given an oracle computing:

$$\mathcal{O}_{b,t}() = \mathrm{MSB}_l(at + b \mod n)$$

with t u.i.d. in $\mathbb{Z}_n^*$, find a.

- Average 1 LZB per signature
- There is noise

Hidden Number Problem (HNP) [1]

Given an oracle computing:

$$\mathcal{O}_{r,s}() = \mathrm{MSB}_l(k \mod n)$$

Exploitation

Hidden Number Problem

- Average 1 LZB per signature
- There is noise

Hidden Number Problem (HNP) [1]

Given an oracle computing:

$$\mathcal{O}_{r,s}() = \mathrm{MSB}_l(xs^{-1}r + H(m)s^{-1} \mod n)$$

find x.

Exploitation

Basic attack [8]

- Collect N signatures, take d of the fastest

Exploitation

Basic attack [8]

- Collect N signatures, take d of the fastest
- Assume some bounds l_i: $|k_i| = |xt_i - u_i| = |xs_i^{-1}r_i + H(m_i)s_i^{-1}| < n/2^{l_i}$

Exploitation

Basic attack [8]

- Collect N signatures, take d of the fastest
- Assume some bounds l_i: $|k_i| = |xt_i - u_i| = |xs_i^{-1}r_i + H(m_i)s_i^{-1}| < n/2^{l_i}$
- Construct a lattice with basis **B** and reduce it:

$$\mathbf{B} = \begin{pmatrix} 2^{l_1}n & 0 & 0 & \dots & 0 & 0 \\ 0 & 2^{l_2}n & 0 & \dots & 0 & 0 \\ & \vdots & & & \vdots & \\ 0 & 0 & 0 & \dots & 2^{l_d}n & 0 \\ 2^{l_1}t_1 & 2^{l_2}t_2 & 2^{l_3}t_3 & \dots & 2^{l_d}t_d & 1 \end{pmatrix}$$

Exploitation

Basic attack [8]

- Collect N signatures, take d of the fastest
- Assume some bounds l_i: $|k_i| = |xt_i - u_i| = |xs_i^{-1}r_i + H(m_i)s_i^{-1}| < n/2^{l_i}$
- Construct a lattice with basis **B** and reduce it:

$$\mathbf{B} = \begin{pmatrix} 2^{l_1}n & 0 & 0 & \dots & 0 & 0 \\ 0 & 2^{l_2}n & 0 & \dots & 0 & 0 \\ & \vdots & & & \vdots & \\ 0 & 0 & 0 & \dots & 2^{l_d}n & 0 \\ 2^{l_1}t_1 & 2^{l_2}t_2 & 2^{l_3}t_3 & \dots & 2^{l_d}t_d & 1 \end{pmatrix}$$

- Construct a target $\mathbf{u} = (2^{l_1}u_1, \dots, 2^{l_d}u_d, 0)$

Exploitation

Basic attack [8]

- Collect N signatures, take d of the fastest
- Assume some bounds l_i: $|k_i| = |xt_i - u_i| = |xs_i^{-1}r_i + H(m_i)s_i^{-1}| < n/2^{l_i}$
- Construct a lattice with basis $\mathbf{B}$ and reduce it:

$$\mathbf{B} = \begin{pmatrix} 2^{l_1}n & 0 & 0 & \dots & 0 & 0 \\ 0 & 2^{l_2}n & 0 & \dots & 0 & 0 \\ & \vdots & & & \vdots & \\ 0 & 0 & 0 & \dots & 2^{l_d}n & 0 \\ 2^{l_1}t_1 & 2^{l_2}t_2 & 2^{l_3}t_3 & \dots & 2^{l_d}t_d & 1 \end{pmatrix}$$

- Construct a target $\mathbf{u} = (2^{l_1}u_1, \dots, 2^{l_d}u_d, 0)$
- Solve CVP($\mathbf{B}, \mathbf{u}$). The closest lattice point is often: $\mathbf{v} = (2^{l_1}t_1x, \dots, 2^{l_d}t_dx, x)$

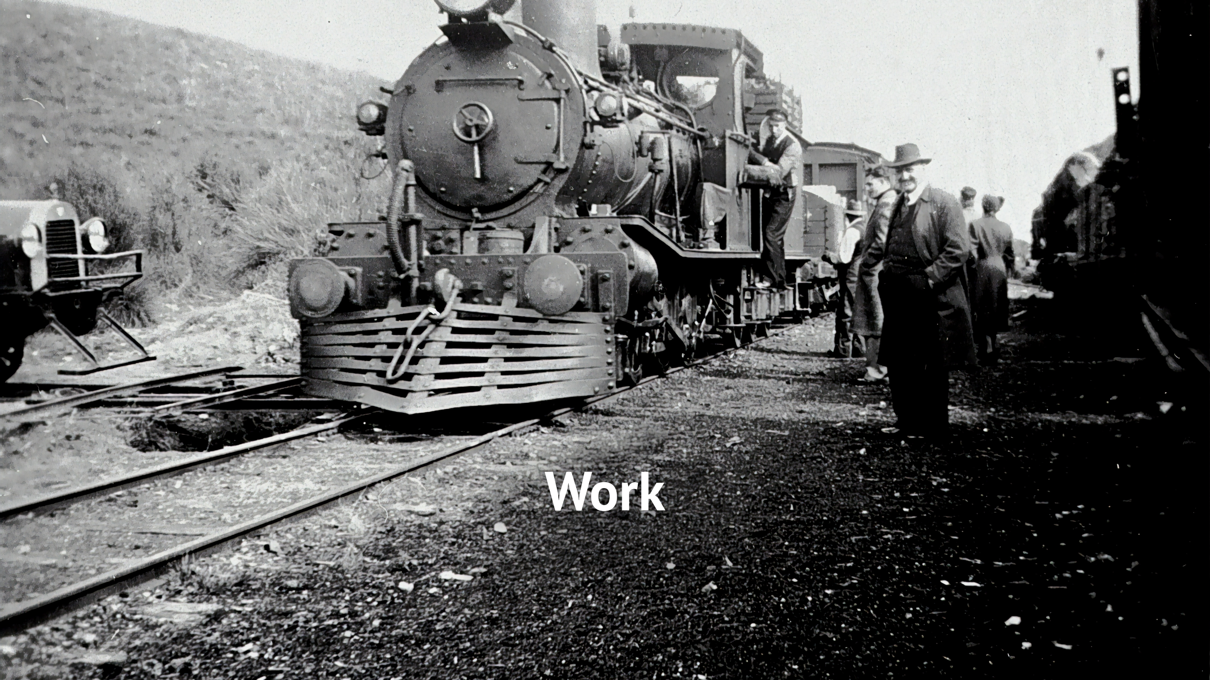
Exploitation

Basic attack [8]

- Collect N signatures, take d of the fastest
- Assume some bounds l_i: $|k_i| = |xt_i - u_i| = |xs_i^{-1}r_i + H(m_i)s_i^{-1}| < n/2^{l_i}$
- Construct a lattice with basis $\mathbf{B}$ and reduce it:

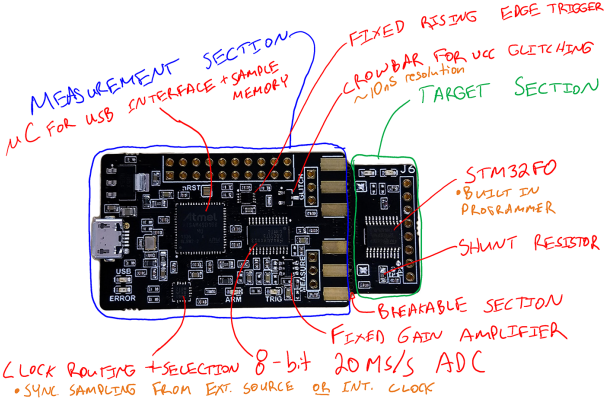
$$\mathbf{B} = \begin{pmatrix} 2^{l_1}n & 0 & 0 & \dots & 0 & 0 \\ 0 & 2^{l_2}n & 0 & \dots & 0 & 0 \\ & \vdots & & & \vdots & \\ 0 & 0 & 0 & \dots & 2^{l_d}n & 0 \\ 2^{l_1}t_1 & 2^{l_2}t_2 & 2^{l_3}t_3 & \dots & 2^{l_d}t_d & 1 \end{pmatrix}$$

- Construct a target $\mathbf{u} = (2^{l_1}u_1, \dots, 2^{l_d}u_d, 0)$
- Solve CVP($\mathbf{B}, \mathbf{u}$). The closest lattice point is often: $\mathbf{v} = (2^{l_1}t_1x, \dots, 2^{l_d}t_dx, x)$
- Because $\forall i$: $(xt_i - u_i) \mod n$ is small


Work

Practical part

You will mount several side-channel attacks on a modified micro-ecc ECDSA implementation running on a ChipWhisperer-Nano board.



pyecsca

Practical part

Prerequisites
Clone the repository https://github.com/J08nY/cwnano-micro-ecc
Have setup ready (see installation.md)

1. Building the target implementation
2. Interacting with the target
3. Running the nonce-reuse attack
4. Running the nonce-bitlength-leak attack
    1. With timing
    2. With powertraces

Practical part

- Work in groups of 2-3, only 19 devices
- Potentially share the devices
- Instructions in tutorial.md
- Consult troubleshooting.md in case of issues

Practical part

Hopeful fix

```
pip install -U ipython pyzmq
```

And reboot VM

Thanks!

J08nY | neuromancer.sk | jan@neuromancer.sk

crocs.fi.muni.cz

Icons from Noun Project & Font Awesome

Photos from Unsplash

References

1. failOverflow; **Console Hacking 2010: PS3 Epic Fail**
2. Joachim Breitner, Nadia Heninger; **Biased Nonce Sense: Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies**
3. Jan Jancar, Vladimir Sedlacek, Petr Svenda, Marek Sys; **Minerva: The curse of ECDSA nonces (Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces)**
4. Martin R. Albrecht, Nadia Heninger; **On Bounded Distance Decoding with Predicate: Breaking the "Lattice Barrier" for the Hidden Number Problem**
5. Chao Sun, Thomas Espitau, Mehdi Tibouchi, Masayuki Abe; **Guessing Bits: Improved Lattice Attacks on (EC)DSA**
6. Daniel Bleichenbacher; **On the generation of one-time keys in DL signature schemes**
7. Dan Boneh, Ramarathnam Venkatesan; **Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes**
8. Billy Bob Brumley, Nicola Tuveri; **Remote Timing Attacks are Still Practical**