Timing and Lattice Attacks on a Remote ECDSA OpenSSL Server: How Practical Are They Really?

David Wong
University of Bordeaux and NCC Group

2003

Remote Timing Attacks are Practical

David Brumley
Stanford University
dbrumley@cs.stanford.edu

Dan Boneh
Stanford University
dabo@cs.stanford.edu

Abstract

Timing attacks are usually used to attack weak computing devices such as smartcards. We show that timing attacks apply to general software systems. Specifically, we devise a timing attack against OpenSSL. Our experiments show that we can extract private keys from an OpenSSL-based web server running on a machine in the local network. Our results demonstrate that timing attacks against network servers are practical and therefore security systems should defend against them.

1 Introduction

Timing attacks enable an attacker to extract secrets

The attacking machine and the server were in different buildings with three routers and multiple switches between them. With this setup we were able to extract the SSL private key from common SSL applications such as a web server (Apache+mod_SSL) and a SSL-tunnel.

Interprocess. We successfully mounted the attack between two processes running on the same machine. A hosting center that hosts two domains on the same machine might give management access to the admins of each domain. Since both domain are hosted on the same machine, one admin could use the attack to extract the secret key belonging to the other domain.

Virtual Machines. A Virtual Machine Monitor (VMM) is often used to enforce isolation between two Virtual Machines (VM) running on the same processor. One could protect an RSA private key by storing it in one VM and enabling other VM's to make

2011

Remote Timing Attacks are Still Practical*

Billy Bob Brumley and Nicola Tuveri

Aalto University School of Science, Finland
{bbrumley,ntuveri}@tcs.hut.fi

Abstract. For over two decades, timing attacks have been an active area of research within applied cryptography. These attacks exploit cryptosystem or protocol implementations that do not run in constant time. When implementing an elliptic curve cryptosystem with a goal to provide side-channel resistance, the scalar multiplication routine is a critical component. In such instances, one attractive method often suggested in the literature is Montgomery's ladder that performs a fixed sequence of curve and field operations. This paper describes a timing attack vulnerability in OpenSSL's ladder implementation for curves over binary fields. We use this vulnerability to steal the private key of a TLS server where the server authenticates with ECDSA signatures. Using the timing of the exchanged messages, the messages themselves, and the signatures, we mount a lattice attack that recovers the private key. Finally, we describe and implement an effective countermeasure.

Keywords: Side-channel attacks, timing attacks, elliptic curve cryptography, lattice attacks.

1 Introduction

Side-channel attacks utilize information leaked during the execution of a protocol. These attacks differ from traditional cryptanalysis attacks since side-

OpenSSL

https://www.openssl.org

OpenSSL

Cryptography and SSL/TLS Toolkit

Search

Home | Downloads | Docs | News | Policies | Community | Support

Welcome to the OpenSSL Project

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols as well as a full-strength general purpose cryptography library. The project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and

Home

Downloads: Source code

Docs: FAQ, FIPS, manpages, ...

News: Latest information

Policies: How we operate

Community: Blog, bugs, email, ...

Support: Commercial support and contracting

Sponsor Acknowledgements

ECDSA

```
Ecdsa-Sig-Value ::= SEQUENCE {
    r        INTEGER,
    s        INTEGER
}
```

From RFC 4492

$$r = ([k]G)_x \bmod n$$
$$s = (h(m) + dr)k^{-1} \bmod n.$$

ECDSA

```
Ecdsa-Sig-Value ::= SEQUENCE {
    r        INTEGER,
    s        INTEGER
}
```

From RFC 4492

$$r = ([k]G)_x \bmod n$$

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

message

ECDSA

```
Ecdsa-Sig-Value ::= SEQUENCE {
    r        INTEGER,
    s        INTEGER
}
```

From RFC 4492

$$r = ([k]G)_x \bmod n$$

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

private key

message

ECDSA

```
Ecdsa-Sig-Value ::= SEQUENCE {
    r        INTEGER,
    s        INTEGER
}
```

From RFC 4492

nonce

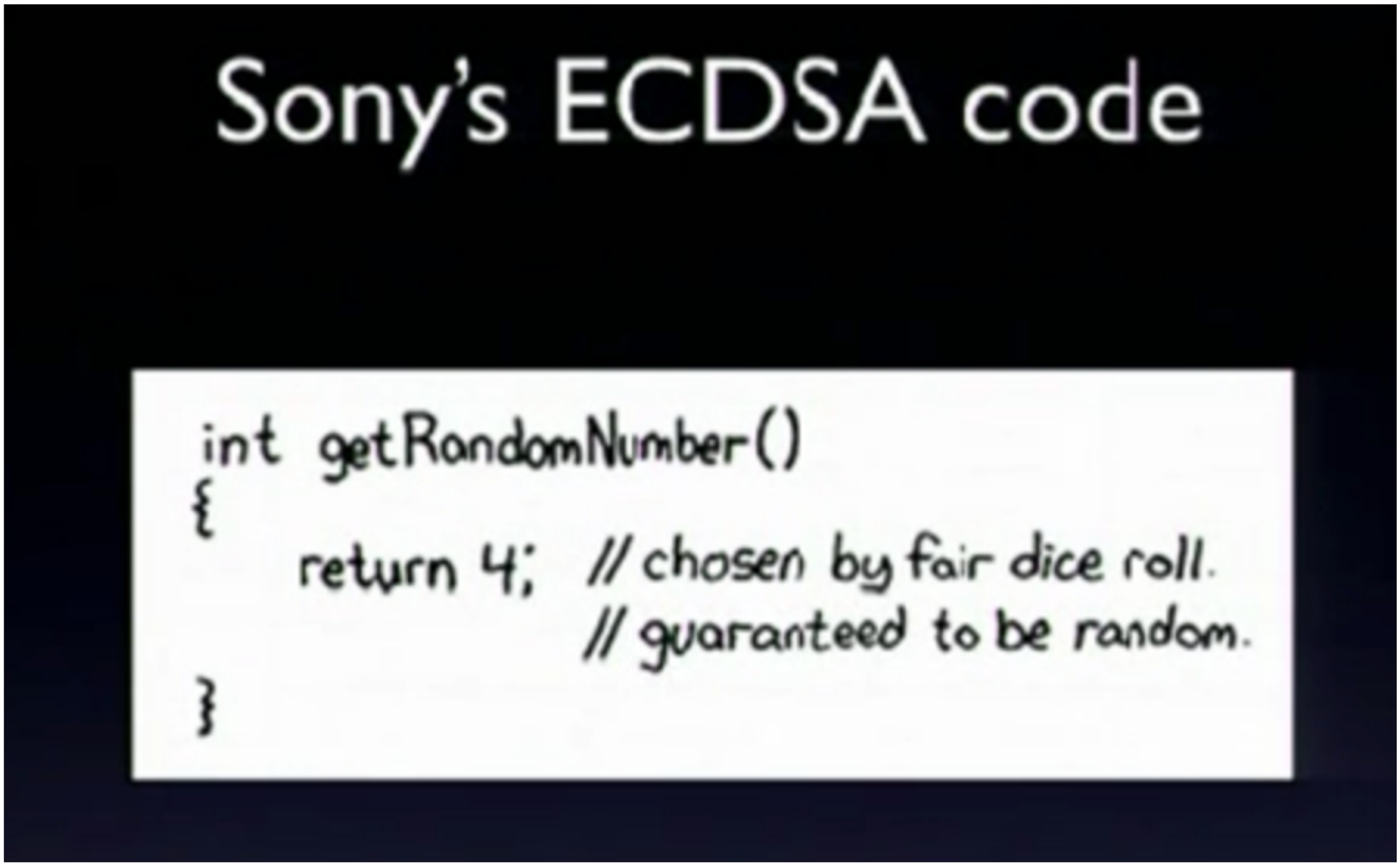
$$r = ([k]G)_x \bmod n$$

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

private key

message

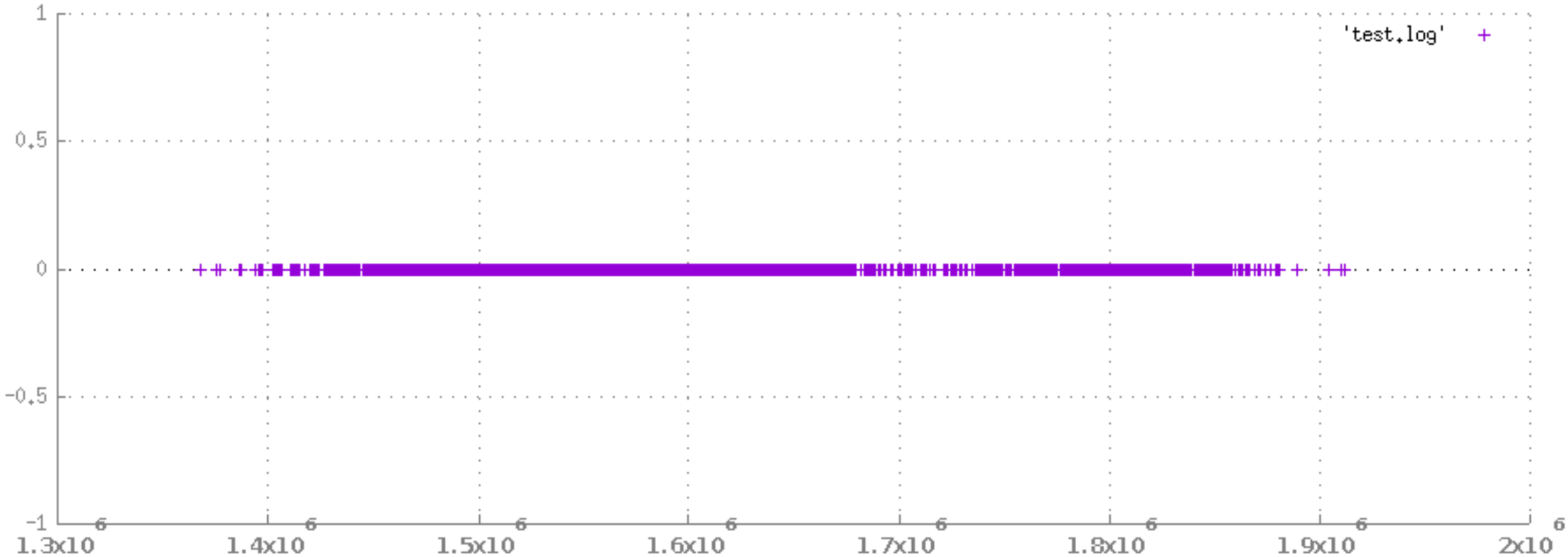
The PS3 Failure



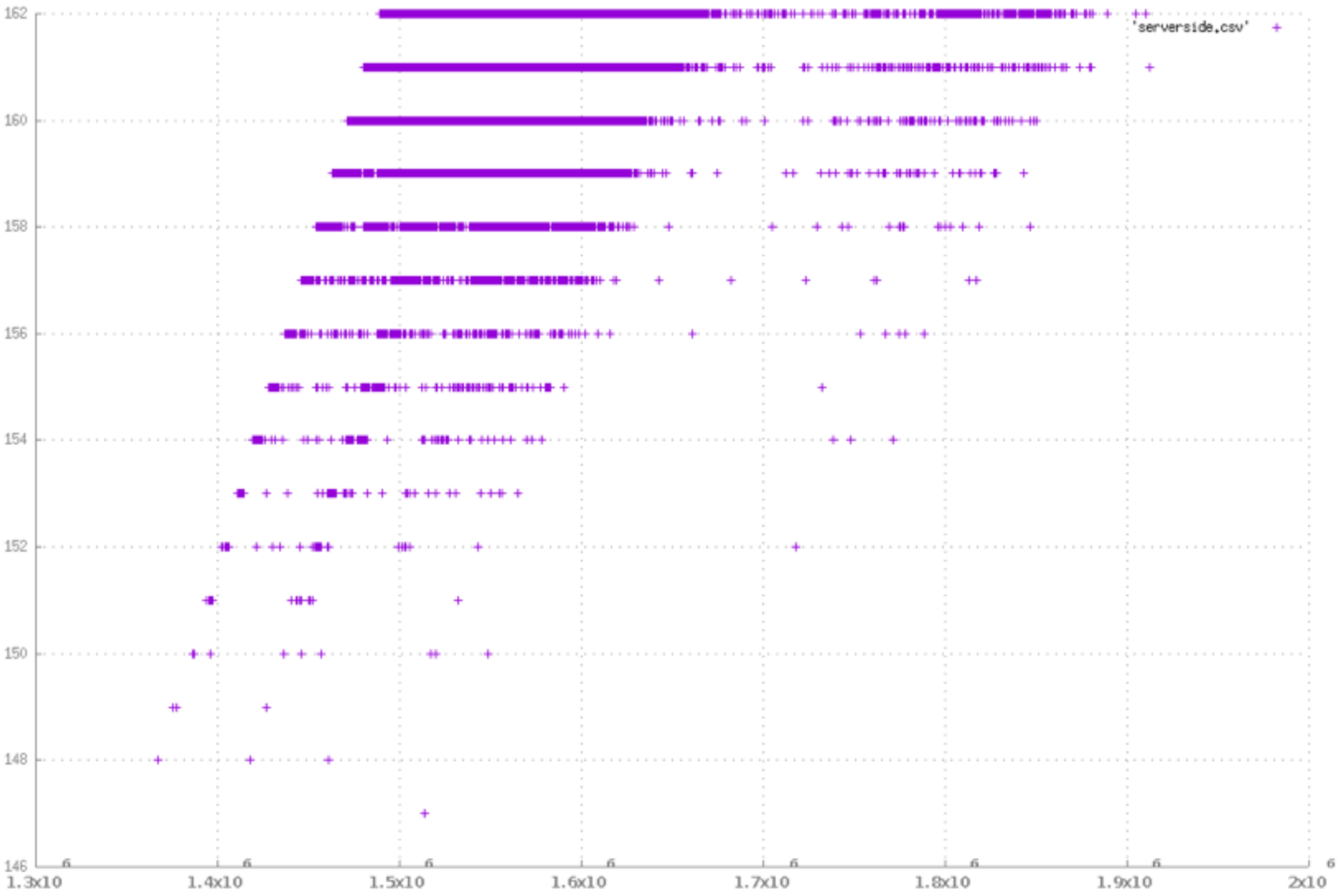
```
/* find top most bit and go one past it */
i = scalar->top - 1;
mask = BN_TBIT;
word = scalar->d[i];
while (!(word & mask)) mask >>= 1;
mask >>= 1;
/* if top most bit was at word break, go to next word */
if (!mask)
        {
        i--;
        mask = BN_TBIT;
        }

for (; i >= 0; i--)
        {
        word = scalar->d[i];
        while (mask)
                {
                BN_consttime_swap(word & mask, x1, x2, group->field.top);
                BN_consttime_swap(word & mask, z1, z2, group->field.top);
                if (!gf2m_Madd(group, &point->X, x2, z2, x1, z1, ctx)) goto err;
                if (!gf2m_Mdouble(group, x1, z1, ctx)) goto err;
                BN_consttime_swap(word & mask, x1, x2, group->field.top);
```

```
/* find top most bit and go one past it */
i = scalar->top - 1;
mask = BN_TBIT;
word = scalar->d[i];
while (!(word & mask)) mask >>= 1;
mask >>= 1;
/* if top most bit was at word break, go to next word */
if (!mask)
        {
        i--;
        mask = BN_TBIT;
        }

for (; i >= 0; i--)
        {
        word = scalar->d[i];
        while (mask)
                {
                BN_consttime_swap(word & mask, x1, x2, group->field.top);
                BN_consttime_swap(word & mask, z1, z2, group->field.top);
                if (!gf2m_Madd(group, &point->X, x2, z2, x1, z1, ctx)) goto err;
                if (!gf2m_Mdouble(group, x1, z1, ctx)) goto err;
                BN_consttime_swap(word & mask, x1, x2, group->field.top);
```

After producing 10,000 signatures



After producing 10,000 signatures



2001

LATTICE ATTACKS ON DIGITAL SIGNATURE SCHEMES

N.A. HOWGRAVE-GRAHAM AND N.P. SMART

ABSTRACT. We describe a lattice attack on the Digital Signature Algorithm (DSA) when used to sign many messages, m_i, under the assumption that a proportion of the bits of each of the associated ephemeral keys, y_i, can be recovered by alternative techniques.

1. INTRODUCTION

Lattice attacks have recently been used to attack RSA schemes under various additional assumptions, such as low exponent versions of RSA, or factoring the modulus when a certain portion of the bits of p are known in advance. Many of these attacks have derived from ground breaking ideas of Coppersmith on how one can use the LLL algorithm [9] to solve univariate and bivariate modular polynomial equations. For more details on this and related matters the reader should consult, [2], [4], [5], [7] and [8].

ElGamal signatures, see [6], are based on the assumption that one has a finite abelian group, G, for which it is computationally infeasible to solve the discrete logarithm and Diffie-Hellman problems. ElGamal type signature schemes have been deployed and standardized in the Digital Signature Algorithm, DSA, and its


TLS Client
• ClientHello

ClientHello

```
▽ Secure Sockets Layer
  ▽ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 302
    ▽ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 298
        Version: TLS 1.2 (0x0303)
      ▷ Random
        Session ID Length: 0
        Cipher Suites Length: 148
      ▽ Cipher Suites (74 suites)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
          Cipher Suite: TLS_DHE_DSS_WITH_AES_256_GCM_SHA384 (0x00a3)
```

TLS Client
• ClientHello
• ServerHello
• ServerCertificate
• ServerHelloDone
TLS Server

TLS Client

• ClientHello

• ServerHello
• ServerCertificate

• ServerHelloDone

TLS Server

TLS Client

• ClientKeyExchange

• ChangeCipherSpec
• Finished

• ChangeCipherSpec
• Finished

TLS Server

TLS Client

• ClientHello

• ServerHello
• ServerCertificate
• ServerKeyExchange

• ServerHelloDone

TLS Server

TLS Client

• ClientKeyExchange

• ChangeCipherSpec
• Finished

• ChangeCipherSpec
• Finished

TLS Server

Wireshark view of the Server Key Exchange

```
▷ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
▷ TLSv1.2 Record Layer: Handshake Protocol: Certificate
▽ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 125
  ▽ Handshake Protocol: Server Key Exchange
      Handshake Type: Server Key Exchange (12)
      Length: 121
    ▽ EC Diffie-Hellman Server Params
        Curve Type: named_curve (0x03)
        Named Curve: secp256r1 (0x0017)
        Pubkey Length: 65
        Pubkey: 040afe0ea00c08b128d883adc07f9da7686faf4c90cc8681...
      ▷ Signature Hash Algorithm: 0x0603
        Signature Length: 48
        Signature: 302e021501eac9ea0463fd96db7cd967f63ccd9af291de46...
▷ TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
```

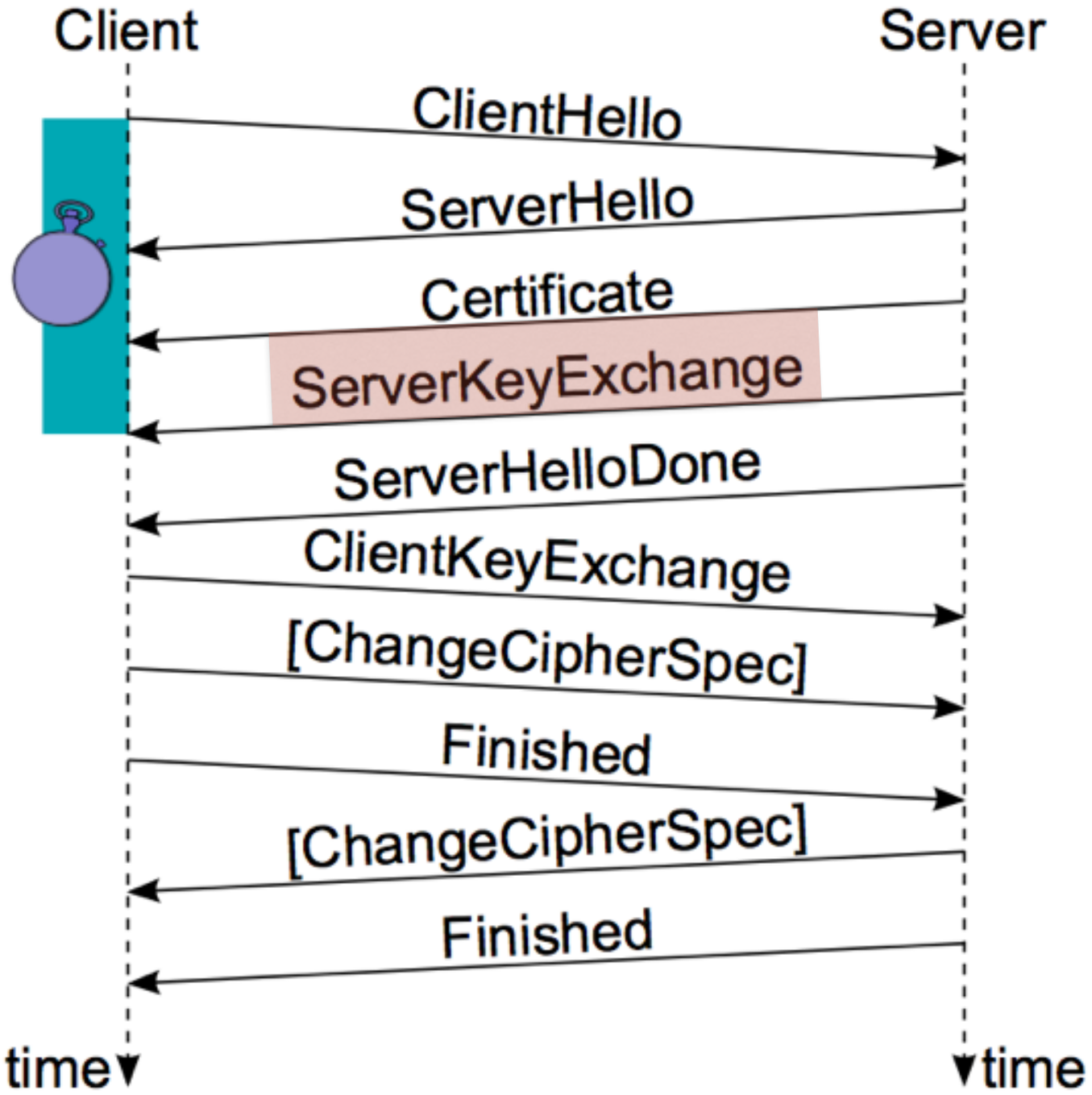
RFC 4492

```
signed_params:   A hash of the params, with the signature appropriate
   to that hash applied.  The private key corresponding to the
   certified public key in the server's Certificate message is used
   for signing.

       enum { ecdsa } SignatureAlgorithm;

   select (SignatureAlgorithm) {
       case ecdsa:
           digitally-signed struct {
               opaque sha_hash[sha_size];
           };
   } Signature;


ServerKeyExchange.signed_params.sha_hash
       SHA(ClientHello.random + ServerHello.random +
                                          ServerKeyExchange.params);
```

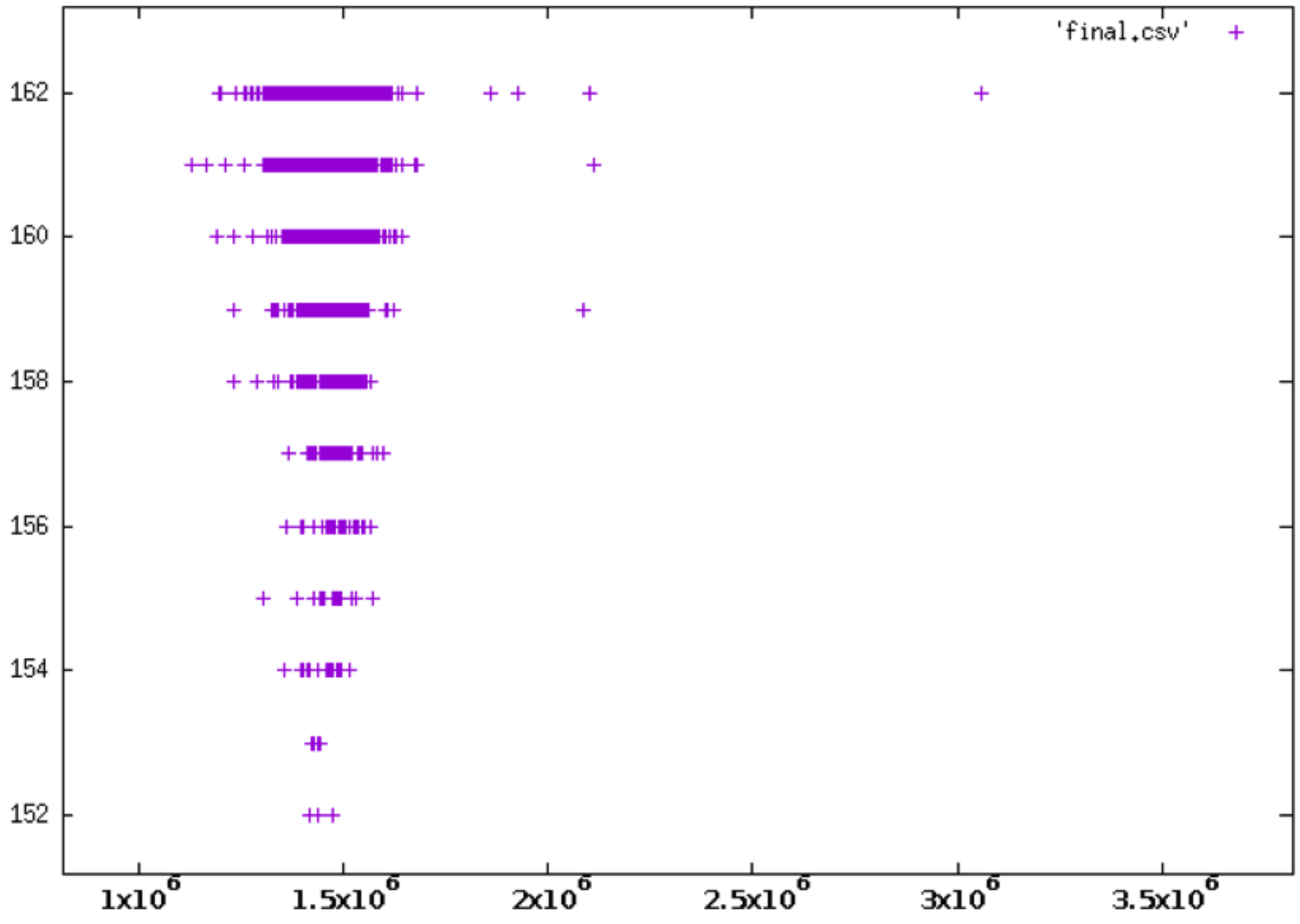
The Timing Attack

```
int main(int argc, char *argv[])
{
  char clienthello[] = "\x16\x03\x01\x01\x2e\x01\x00\x01\x2a\x03\x03\x6d\x70\xd7\xa7\x43\x44\xe7\xcc\xf7\xc3\xac\xe7\xc7\x7f\x\
f3\x9f\x9d\x9b\x2e\xa5\x6d\x26\xac\x92\x92\x22\x4a\xa3\x2f\x17\xc1\x0b\x00\x00\x94\xc0\x30\xc0\x2c\xc0\x28\xc0\x24\xc0\x14\xc0\x0a
00\xa3\x00\x9f\x00\x6b\x00\x6a\x00\x39\x00\x38\x00\x88\x00\x87\xc0\x32\xc0\x2e\xc0\x2a\xc0\x26\xc0\x0f\xc0\x05\x00\x9d\x00\x3d\x00
35\x00\x84\xc0\x2f\xc0\x2b\xc0\x27\xc0\x23\xc0\x13\xc0\x09\x00\xa2\x00\x9e\x00\x67\x00\x40\x00\x33\x00\x32\x00\x9a\x00\x99\x00\x45
00\x44\xc0\x31\xc0\x2d\xc0\x29\xc0\x25\xc0\x0e\xc0\x04\x00\x9c\x00\x3c\x00\x2f\x00\x96\x00\x41\x00\x07\xc0\x11\xc0\x07\xc0\x0c\xc0
02\x00\x05\x00\x04\xc0\x12\xc0\x08\x00\x16\x00\x13\xc0\x0d\xc0\x03\x00\x0a\x00\x15\x00\x12\x00\x09\x00\x14\x00\x11\x00\x08\x00\x06
00\x03\x00\xff\x01\x00\x00\x6d\x00\x0b\x00\x04\x03\x00\x01\x02\x00\x0a\x00\x34\x00\x32\x00\x0e\x00\x0d\x00\x19\x00\x0b\x00\x0c\x00
18\x00\x09\x00\x0a\x00\x16\x00\x17\x00\x08\x00\x06\x00\x07\x00\x14\x00\x15\x00\x04\x00\x05\x00\x12\x00\x13\x00\x01\x00\x02\x00\x03
00\x0f\x00\x10\x00\x11\x00\x23\x00\x00\x00\x0d\x00\x20\x00\x1e\x06\x01\x06\x02\x06\x03\x05\x01\x05\x02\x05\x03\x04\x01\x04\x02\x04
03\x03\x01\x03\x02\x03\x03\x02\x01\x02\x02\x02\x03\x00\x0f\x00\x01\x01";

  int iteration = atoi(argv[1]);

  uint64_t cycles;
  unsigned int ii;

  for(ii = 0; ii < iteration; ii++){
    init();
    printf("#%i\n", ii);
    cycles = send_request(ii, "10.75.77.60", 4433, clienthello, 307);
  }

  return 0;
}
```

Timing a Round-Trip



From a remote machine



Wanna Help?

moi@davidwong.fr

2001

LATTICE ATTACKS ON DIGITAL SIGNATURE SCHEMES

N.A. HOWGRAVE-GRAHAM AND N.P. SMART

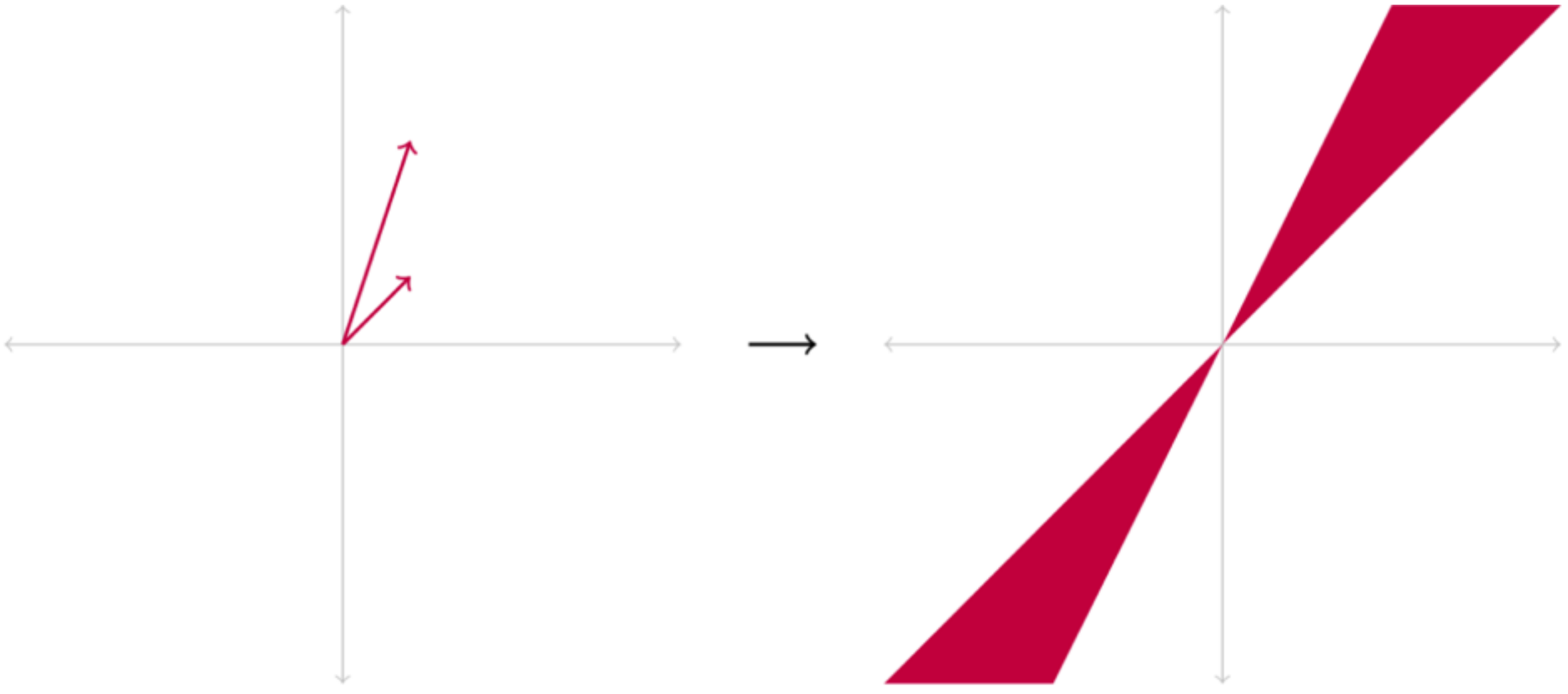
ABSTRACT. We describe a lattice attack on the Digital Signature Algorithm (DSA) when used to sign many messages, m_i, under the assumption that a proportion of the bits of each of the associated ephemeral keys, y_i, can be recovered by alternative techniques.

1. INTRODUCTION

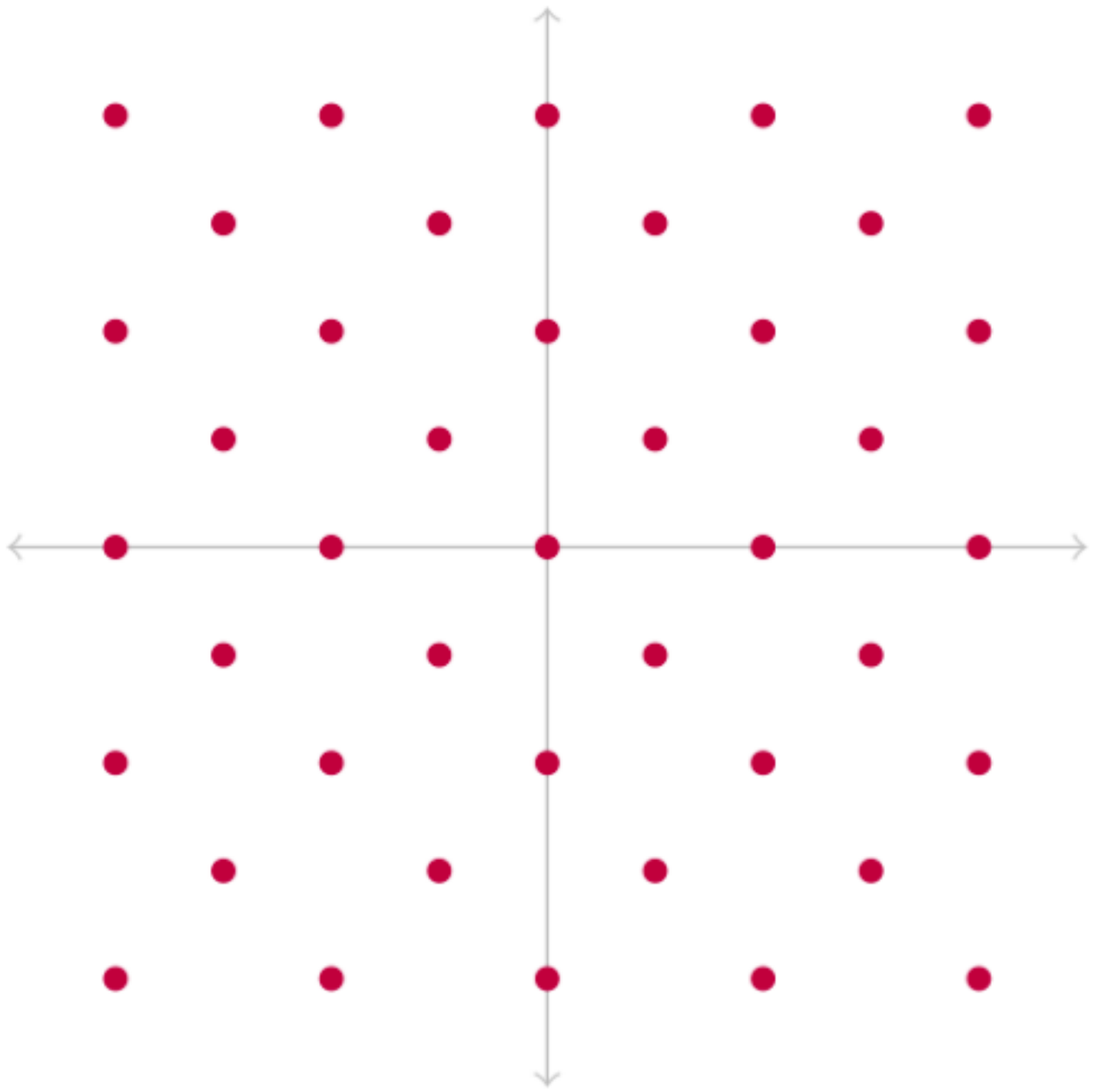
Lattice attacks have recently been used to attack RSA schemes under various additional assumptions, such as low exponent versions of RSA, or factoring the modulus when a certain portion of the bits of p are known in advance. Many of these attacks have derived from ground breaking ideas of Coppersmith on how one can use the LLL algorithm [9] to solve univariate and bivariate modular polynomial equations. For more details on this and related matters the reader should consult, [2], [4], [5], [7] and [8].

ElGamal signatures, see [6], are based on the assumption that one has a finite abelian group, G, for which it is computationally infeasible to solve the discrete logarithm and Diffie-Hellman problems. ElGamal type signature schemes have been deployed and standardized in the Digital Signature Algorithm, DSA, and its

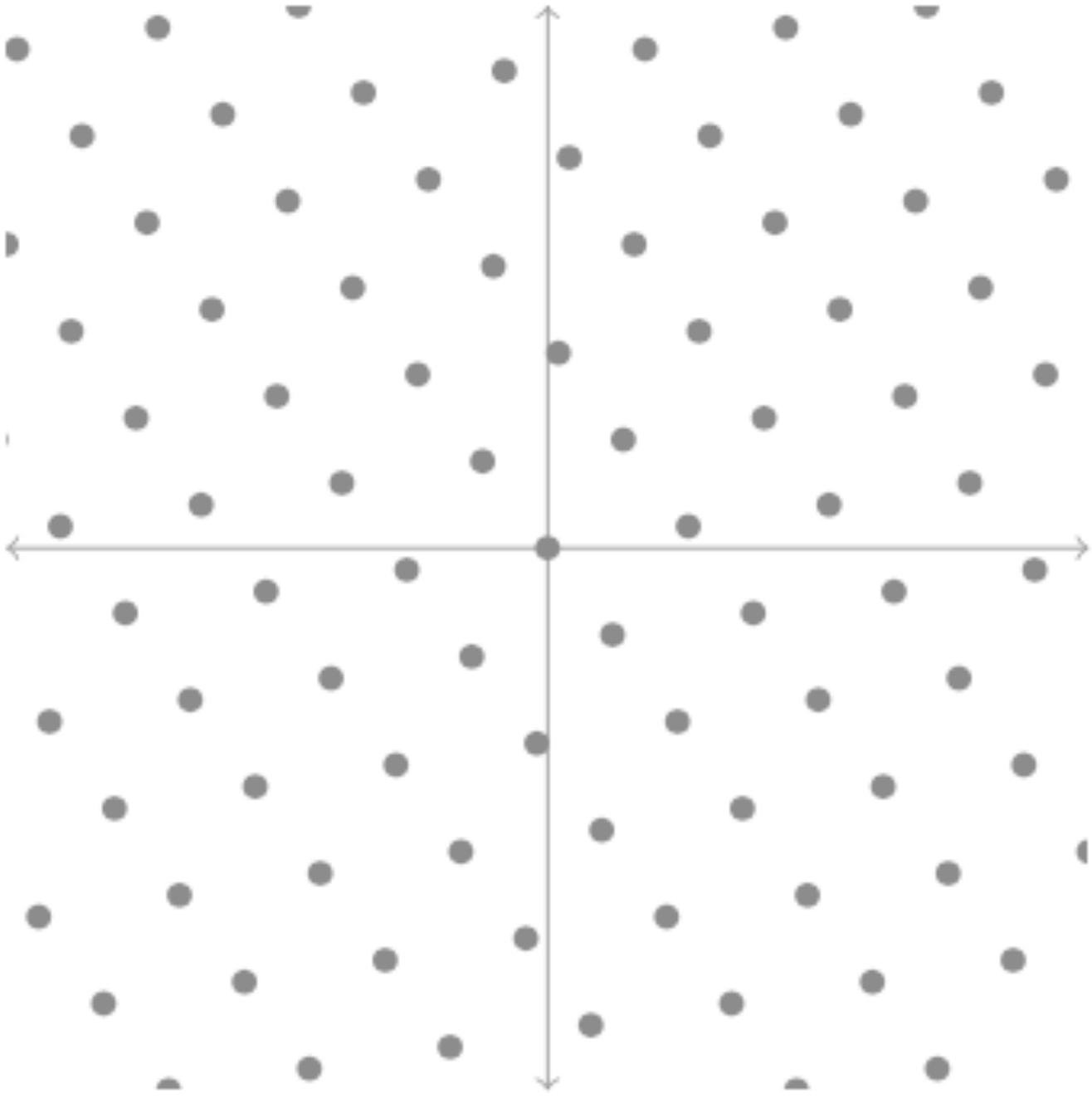
Vector Spaces



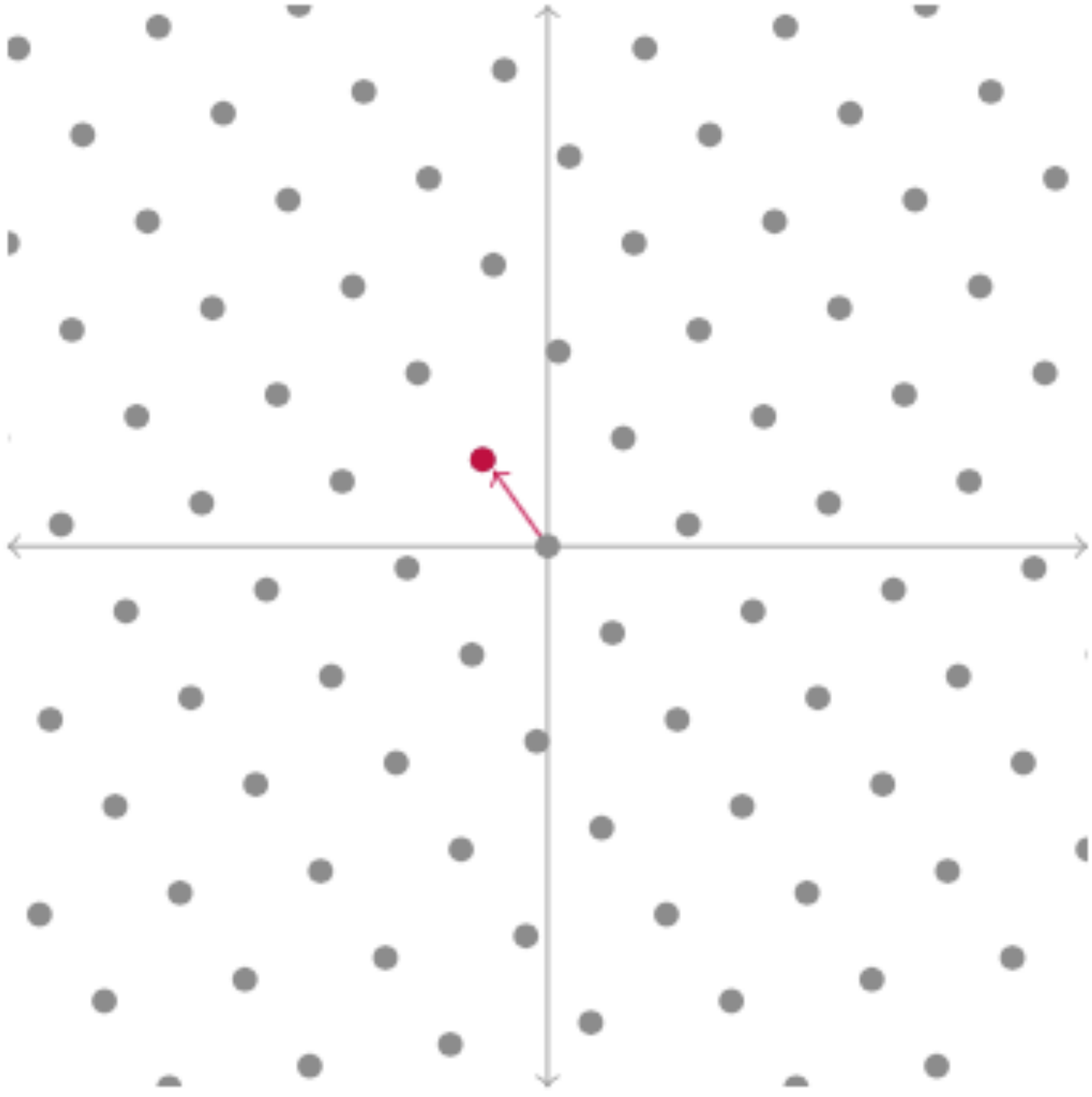
Lattices



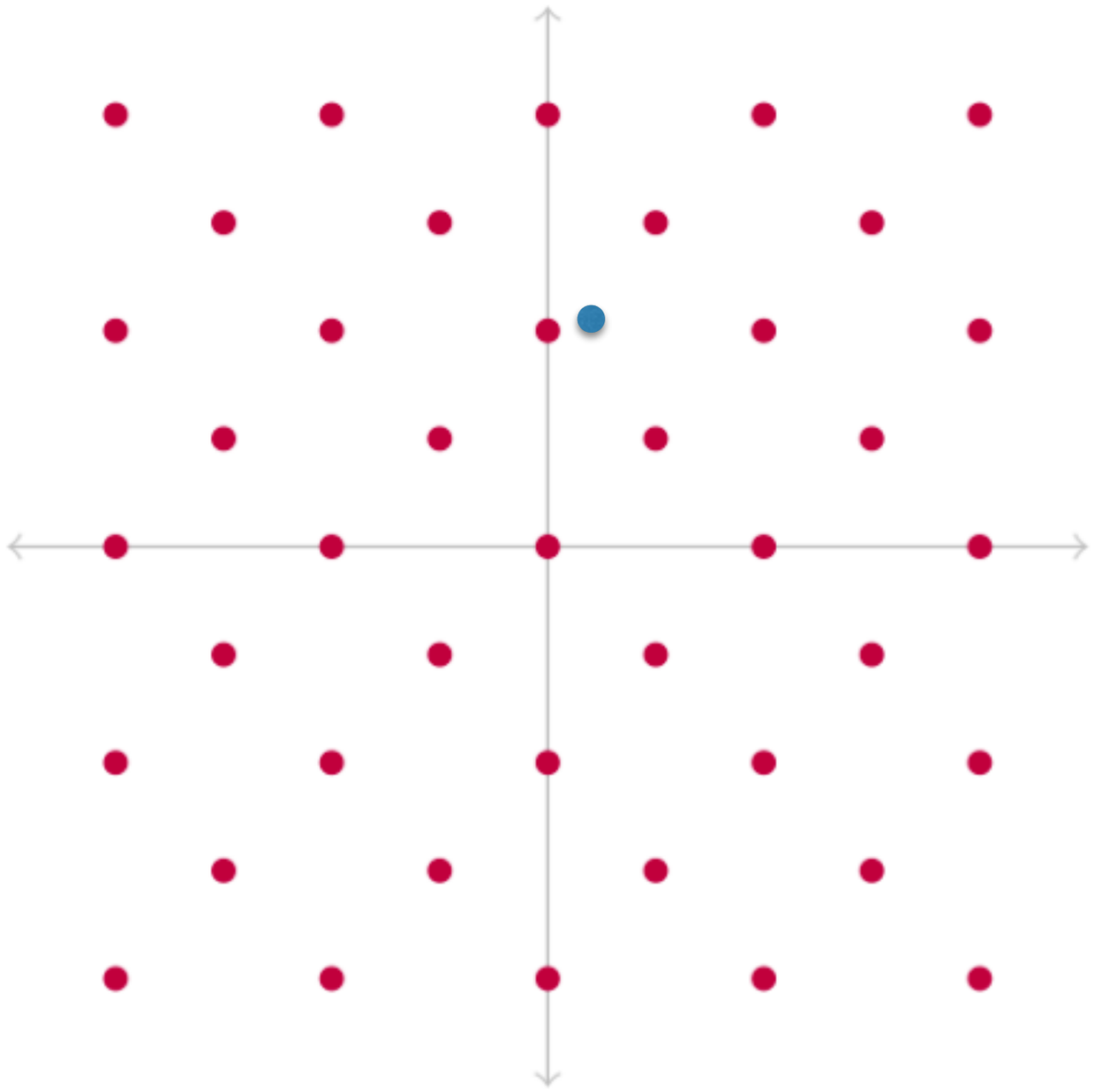
Shortest Vector Problem



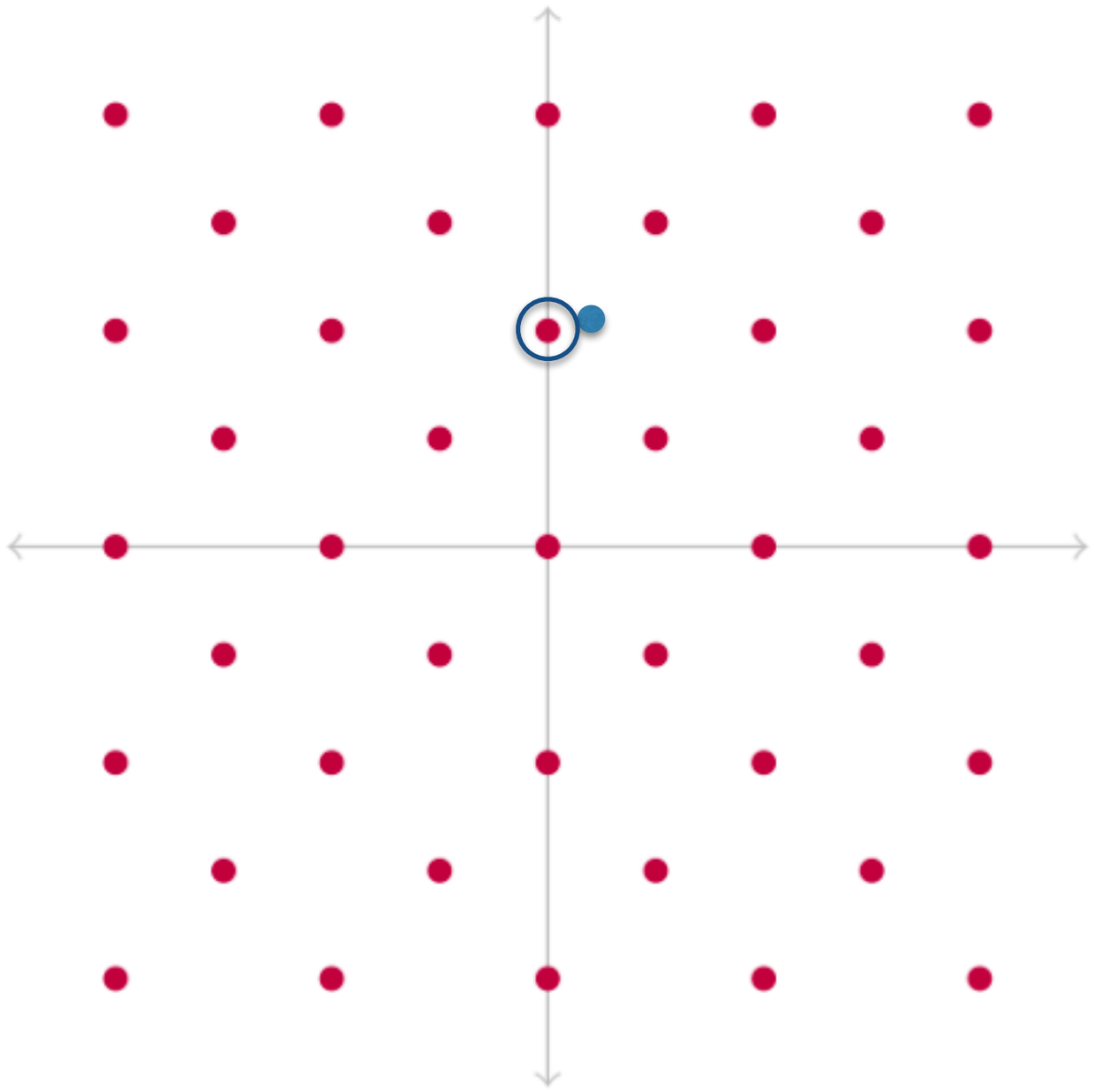
Shortest Vector Problem



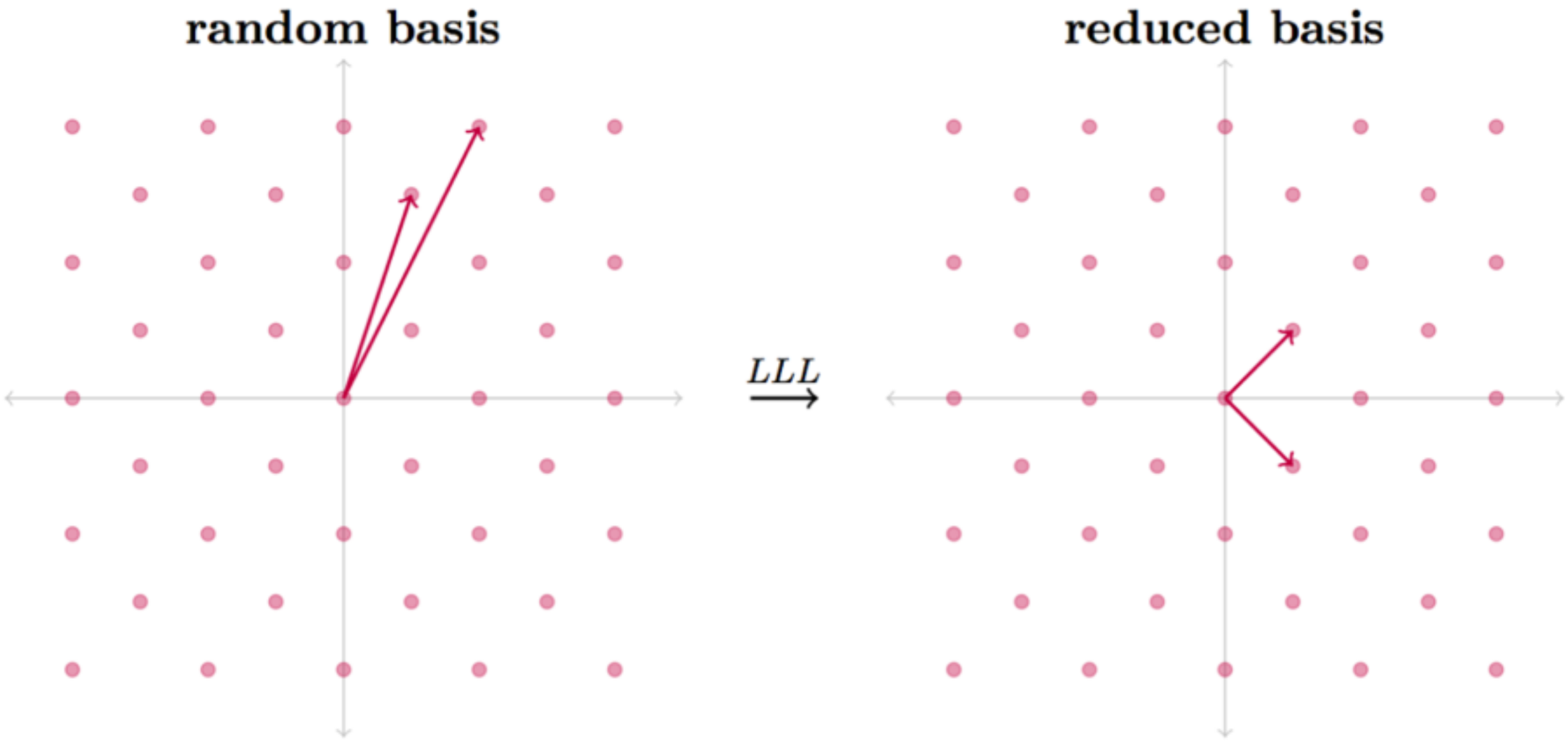
Closest Vector Problem



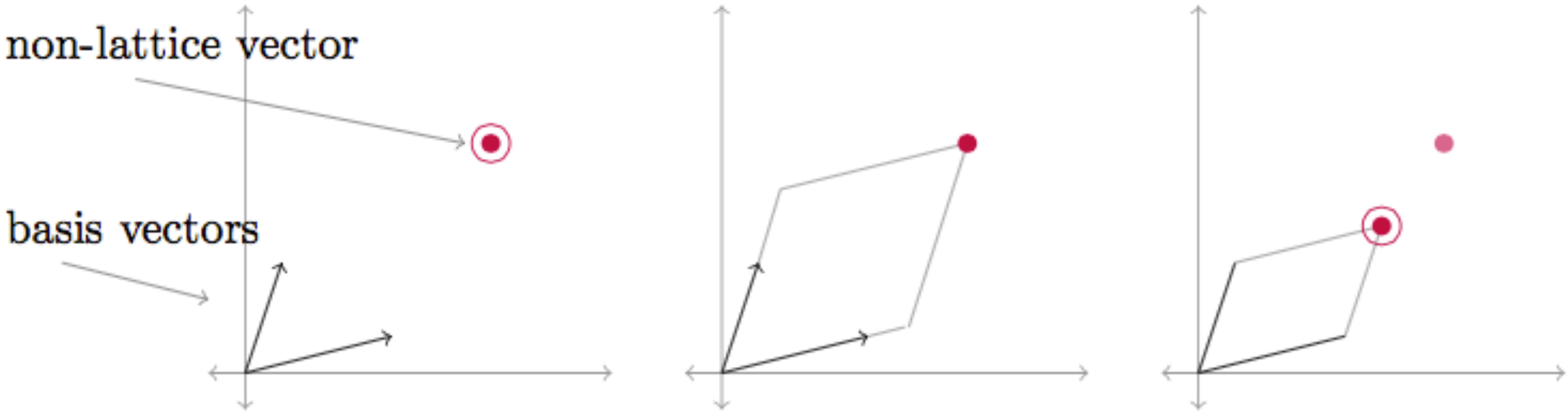
Closest Vector Problem



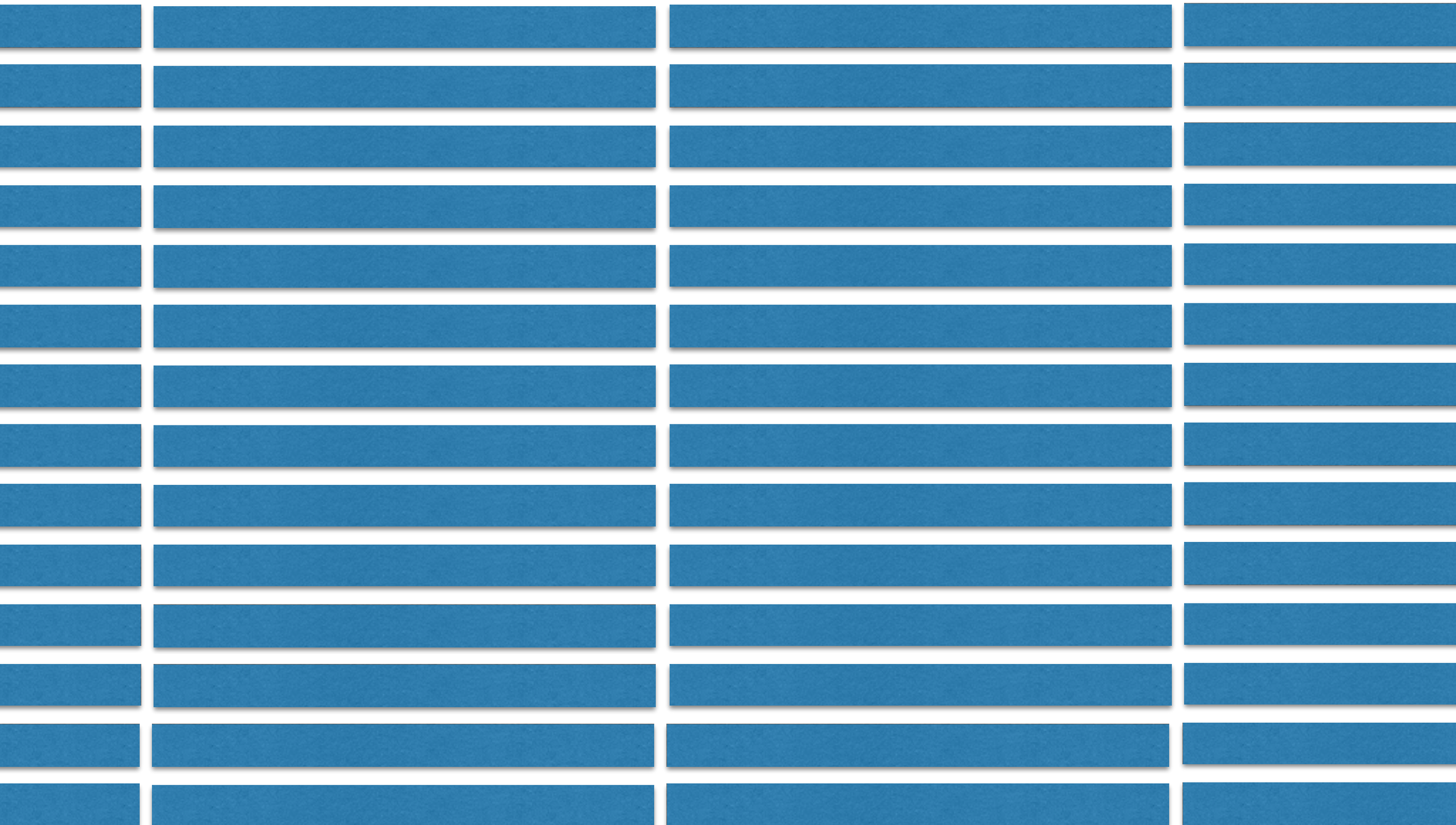
LLL



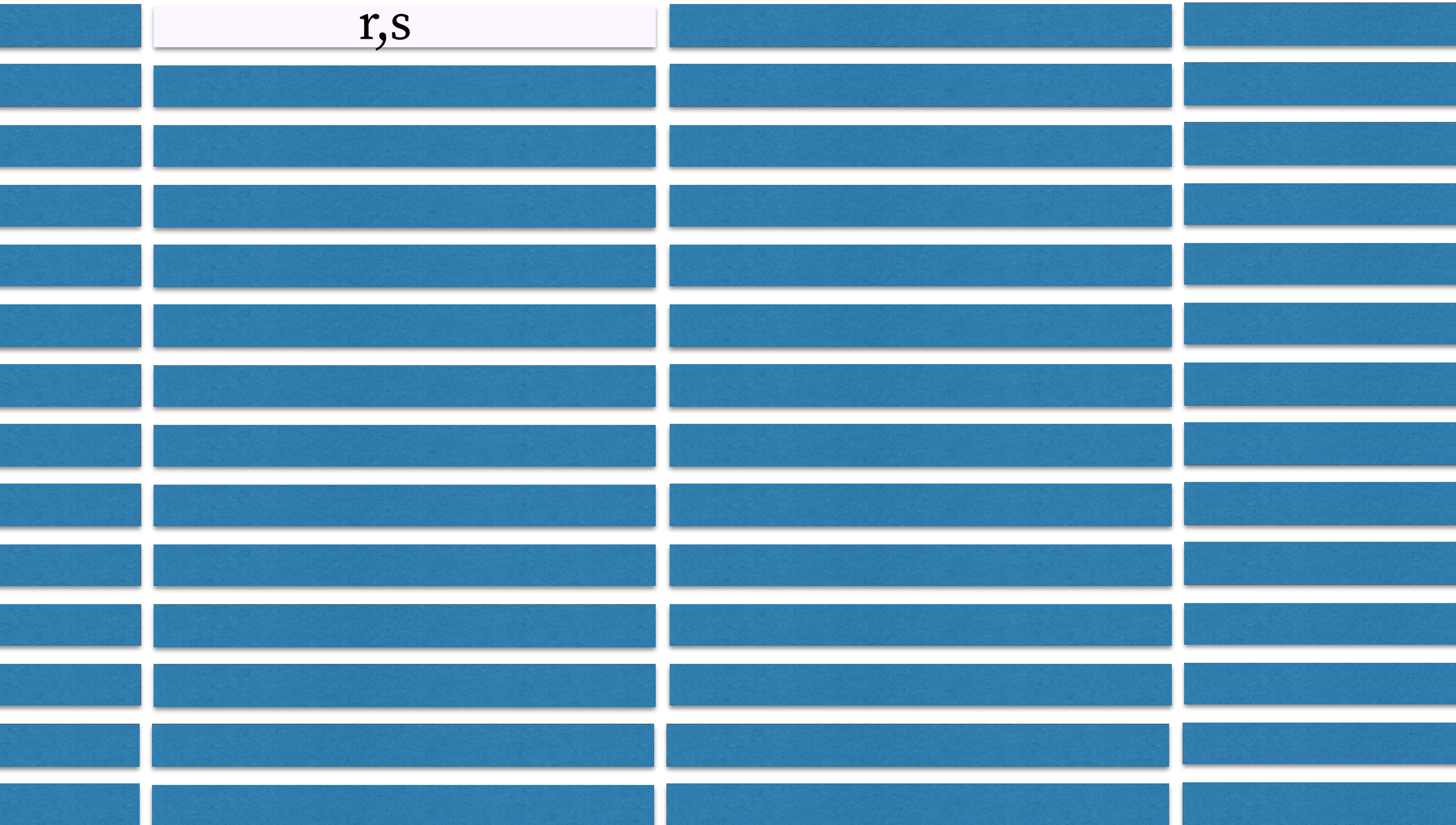
Babai



Reducing our problem to CVP



Reducing our problem to CVP



Reducing our problem to CVP

$$r = ([k]G)_x \bmod n \tag{1}$$

$$s = (h(m) + dr)k^{-1} \bmod n. \tag{2}$$

Reducing our problem to CVP

$$r = ([k]G)_x \bmod n \tag{1}$$
$$s = (h(m) + dr)k^{-1} \bmod n. \tag{2}$$

signatures:

$$\mathrm{h}(m)_i - s_i k_i + dr_i \equiv 0 \pmod{n}$$

Reducing our problem to CVP

$$r = ([k]G)_x \bmod n \tag{1}$$
$$s = (h(m) + dr)k^{-1} \bmod n. \tag{2}$$

signatures:

$$\mathrm{h}(m)_i - s_i k_i + d r_i \equiv 0 \pmod{n}$$

removing d:

$$k_i + A_i k_j + B_i \equiv 0 \pmod{n}$$

$$\begin{cases} k_0 \\ k_1 = -A_1 k_0 + z_1 q - B_1 \\ \cdots \\ k_{n-1} = -A_{n-1} k_0 + z_{n-1} q - B_{n-1} \end{cases}$$

$$
\begin{cases}
k_0 \\
k_1 = -A_1 k_0 + z_1 q - B_1 \\
\dots \\
k_{n-1} = -A_{n-1} k_0 + z_{n-1} q - B_{n-1}
\end{cases}
$$

$$
\begin{pmatrix} k_0 \\ k_1 \\ \vdots \\ k_{n-1} \end{pmatrix} = \begin{pmatrix} -1 & & & \\ A_1 & q & & \\ \vdots & & \ddots & \\ A_{n-1} & & & q \end{pmatrix} \begin{pmatrix} -k_0 \\ z_1 \\ \vdots \\ z_{n-1} \end{pmatrix} - \begin{pmatrix} 0 \\ B_1 \\ \vdots \\ B_{n-1} \end{pmatrix}
$$

DEMO

So What?

- Timing Attacks are everywhere
- Small amount of Info might break your crypto (Lattices!)
- Remote Timing Attacks are hard

Want to try?

https://github.com/mimoo/timing_attack_ecdsa_tls

More about Crypto ?

http://www.cryptologie.net/