Diffie–Hellman, discrete logarithm computation

Inria Nancy, France

Summer school CIMPA, Douala, Cameroon, July 2024

These slides at https://people.rennes.inria.fr/Aurore.Guillevic/talks/2024-07-Douala/24-07-Douala-DL.pdf

Outline

Outline

Introduction: public-key cryptography

Introduced in 1976 (Diffie–Hellman, DH) and 1977 (Rivest–Shamir–Adleman, RSA)
Asymmetric means distinct public and private keys

- encryption with a public key
- decryption with a private key
- deducing the private key from the public key is a very hard problem

Two hard problems:

- Integer factorization (for RSA)
- Discrete logarithm computation in a finite group (for Diffie–Hellman)

Public-key encryption

Alice **Bob**

Public-key encryption

Alice

Bob

public key PK_A

secret key sk_A

Public-key encryption

Alice

Bob

public key PK_A

secret key sk_A

PK_A →

Public-key encryption

Alice

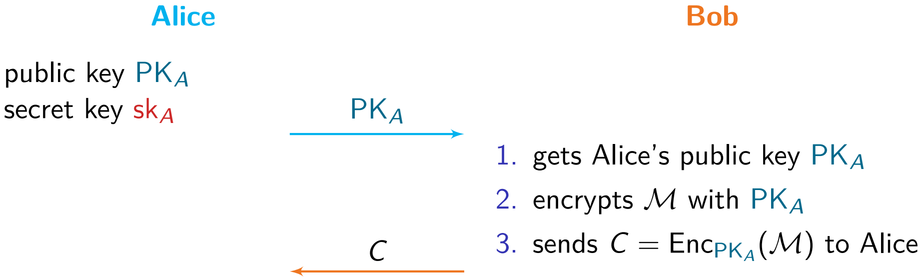
public key PK_A
secret key sk_A

PK_A →

Bob

1. gets Alice's public key PK_A
2. encrypts $\mathcal{M}$ with PK_A
3. sends $C = \mathsf{Enc}_{\mathsf{PK}_A}(\mathcal{M})$ to Alice

Public-key encryption



Public-key encryption

Alice

public key PK_A
secret key sk_A

PK_A →

Bob

1. gets Alice's public key PK_A
2. encrypts $\mathcal{M}$ with PK_A
3. sends $C = \mathsf{Enc}_{\mathsf{PK}_A}(\mathcal{M})$ to Alice

← C

4. gets C from Bob
5. computes $\mathsf{Dec}_{\mathsf{sk}_A}(C) = \mathcal{M}$

Discrete logarithm problem

$\mathbf{G}$ multiplicative group of order q
g generator, $\mathbf{G} = \{1, g, g^2, g^3, \ldots, g^{q-2}, g^{q-1}\}$

Given $h \in \mathbf{G}$, find integer $x \in \{0, 1, \ldots, q-1\}$ such that $h = g^x$.
Exponentiation easy: $(g, x) \mapsto g^x$
Discrete logarithm hard in well-chosen groups $\mathbf{G}$

Choice of group

Prime finite field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ where p is a prime integer
Multiplicative group: $\mathbb{F}_p^* = \{1, 2, \ldots, p-1\}$ (zero omitted)
Multiplication *modulo p*

Finite field $\mathbb{F}_{2^n} = \mathrm{GF}(2^n)$, $\mathbb{F}_{3^m} = \mathrm{GF}(3^m)$ for efficient arithmetic, now broken

Elliptic curves $E\colon y^2 = x^3 + ax + b/\mathbb{F}_p$, $E_a\colon y^2 + xy = x^3 + ax^2 + 1/\mathbb{F}_{2^n}$

Diffie-Hellman key exchange

Alice **Bob**

Diffie-Hellman key exchange

Alice		**Bob**
$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$	public parameters	$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$

Diffie-Hellman key exchange

Alice		**Bob**
$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$	public parameters	$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$
secret key $\mathsf{sk}_A = a \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$		secret key $\mathsf{sk}_B = b \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$
public value $\mathsf{PK}_A = g^a$		public value $\mathsf{PK}_B = g^b$

Diffie-Hellman key exchange

Alice		**Bob**
$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$	public parameters	$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$
secret key $\mathsf{sk}_A = a \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$		secret key $\mathsf{sk}_B = b \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$
public value $\mathsf{PK}_A = g^a$		public value $\mathsf{PK}_B = g^b$

$$\underset{\mathsf{PK}_A}{\overset{\mathsf{PK}_B}{\longleftrightarrow}}$$

Diffie-Hellman key exchange

Alice		**Bob**
$(\mathbf{G},\cdot), g, r = \#\mathbf{G}$	public parameters	$(\mathbf{G},\cdot), g, r = \#\mathbf{G}$
secret key $\mathrm{sk}_A = a \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$		secret key $\mathrm{sk}_B = b \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$
public value $\mathrm{PK}_A = g^a$		public value $\mathrm{PK}_B = g^b$
	PK_B ⟷ PK_A	
gets Bob's public key PK_B		gets Alice's public key PK_A
$sk = \mathrm{PK}_B{}^a = g^{ab}$		$sk = \mathrm{PK}_A{}^b = g^{ab}$

Diffie-Hellman key exchange

Alice		**Bob**
$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$	public parameters	$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$
secret key $\mathrm{sk}_A = a \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$		secret key $\mathrm{sk}_B = b \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$
public value $\mathrm{PK}_A = g^a$		public value $\mathrm{PK}_B = g^b$
	PK_B ⟷ PK_A	
gets Bob's public key PK_B		gets Alice's public key PK_A
$sk = \mathrm{PK}_B{}^a = g^{ab}$		$sk = \mathrm{PK}_A{}^b = g^{ab}$

it works because $(g^a)^b = (g^b)^a = g^{ab}$

Signatures: ElGamal, Schnorr, DSA

With a group **G** of a finite field $\mathbb{F}_p$

- ElGamal signature scheme
- Schnorr signature, patented until February 2008
- Digital Signature Algorithm (DSA)

With a group **G** of an elliptic curve over a finite field

- ECDSA (elliptic curve DSA)
- EdDSA (Edwards curve DSA) since NIST FIPS 186-5 (Feb. 2023)

ElGamal encryption

Alice **Bob**

ElGamal encryption

Alice		**Bob**
$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$	public parameters	$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$

ElGamal encryption

Alice

$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$

secret key $\mathsf{sk}_A = a \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$
public key $\mathsf{PK}_A = g^a$

Bob

$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$

ElGamal encryption

Alice | **Bob**

$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$ | $(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$

secret key $\mathsf{sk}_A = a \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$

public key $\mathsf{PK}_A = g^a$ $\xrightarrow{\mathsf{PK}_A}$

ElGamal encryption

Alice

$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$

secret key $\mathsf{sk}_A = a \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$

public key $\mathsf{PK}_A = g^a$

PK_A →

Bob

$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$

Encryption

1. gets Alice's public key PK_A
2. $\mathcal{M} \in \mathbf{G}$
3. $k_e \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$ at random
4. $\gamma = g^{k_e}$
5. $\mathsf{Enc}_{\mathsf{PK}_A}(\mathcal{M}) = \mathcal{M} \cdot {\mathsf{PK}_A}^{k_e} = \delta$
6. sends $C = (\gamma, \delta)$ to Alice

ElGamal encryption

Alice

$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$

secret key $\mathsf{sk}_A = a \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$
public key $\mathsf{PK}_A = g^a$

$\xrightarrow{\mathsf{PK}_A}$

Bob

$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$

Encryption

1. gets Alice's public key PK_A
2. $\mathcal{M} \in \mathbf{G}$
3. $k_e \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$ at random
4. $\gamma = g^{k_e}$
5. $\mathsf{Enc}_{\mathsf{PK}_A}(\mathcal{M}) = \mathcal{M} \cdot {\mathsf{PK}_A}^{k_e} = \delta$
6. sends $C = (\gamma, \delta)$ to Alice

$\xleftarrow{C}$

ElGamal encryption

Alice

$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$

secret key $\mathrm{sk}_A = a \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$

public key $\mathrm{PK}_A = g^a$

PK_A →

Bob

$(\mathbf{G}, \cdot), g, r = \#\mathbf{G}$

Encryption

1. gets Alice's public key PK_A
2. $\mathcal{M} \in \mathbf{G}$
3. $k_e \leftarrow (\mathbb{Z}/r\mathbb{Z})^*$ at random
4. $\gamma = g^{k_e}$
5. $\mathrm{Enc}_{\mathrm{PK}_A}(\mathcal{M}) = \mathcal{M} \cdot {\mathrm{PK}_A}^{k_e} = \delta$
6. sends $C = (\gamma, \delta)$ to Alice

← C

Decryption

7. get $C = (\gamma, \delta)$ from Bob
8. $\mathrm{Dec}_{\mathrm{sk}_A}(C) = (\gamma^{-a}) \cdot \delta = \mathcal{M}$

Choosing key sizes

Symmetric ciphers (AES): key sizes are 128, 192 or 256 bits.
Perfect symmetric cipher: trying all keys of size n bits takes 2^n tests
$\rightarrow$ **brute-force search**

perfect symmetric cipher with secret key in $[0, 2^n - 1]$, of n bits $\leftrightarrow$ n bits of security

For a Diffie-Hellman group $\mathbf{G}$ over a prime field $\mathbb{F}_p$:
n bits of security $\leftrightarrow$ the best (mathematical) attack to solve a DH instance in $\mathbf{G}$ should take at least 2^n steps

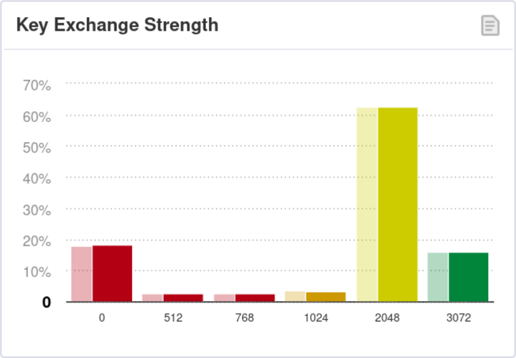
- what is the fastest attack?
- how much time does it take with respect to the size $\#\mathbf{G}$ of $\mathbf{G}$ and the representation of $\mathbf{G}$?

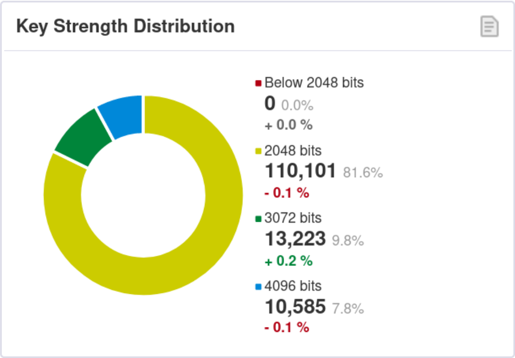
Diffie-Hellman over a prime field has much larger key sizes compared to a symmetric cipher.
Cipher suite: a pair of symmetric and asymmetric ciphers offering the same level of security.

Examples

https://www.lemonde.fr/, https, security information →
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, 128 bits, TLS 1.2

Qualys. SSL Labs https://www.ssllabs.com/ssl-pulse/





Particles

n	2^n	Examples
32	$2^{32} = 10^{9.6}$	number of humans on Earth
47	$2^{47} = 10^{14.2}$	distance Earth - Sun in millimeters ($149.6 \cdot 10^{12}$) number of operations in one day on a processor at 2 GHz
56	$2^{55.8} = 10^{16.8}$	number of operations in one year on a processor at 2 GHz
79	$2^{79} = 10^{23.8}$	Avogadro number: atoms of Carbon 12 in 1 mol
82	$2^{82.3} = 10^{24.8}$	mass of Earth in kilogrammes
100	$2^{100} = 10^{30}$	number of operations in $13.77 \cdot 10^9$ years (age of the universe) on a processor at 2 GHz
155	$2^{155} = 10^{46.7}$	number of molecules of water on Earth
256	$2^{256} = 10^{77.1}$	number of electrons in universe

Courtesy Marine Minier

Boiling water

Universal Security; From bits and mips to pools, lakes – and beyond
Arjen Lenstra, Thorsten Kleinjung, and Emmanuel Thomé
https://hal.inria.fr/hal-00925622

- 2^{90} operations require enough energy to boil the lake of Genève
- 2^{114} operations: boiling all the water on Earth
- 2^{128} operations: boiling 16,000 planets like the Earth

Outline

Asymmetric cryptography

Factorization (RSA cryptosystem)

Discrete logarithm problem (use in Diffie-Hellman, etc)

Given a finite cyclic group $(\mathbf{G}, \cdot)$, a generator g and $h \in \mathbf{G}$, compute x s.t. $h = g^x$.
$\rightarrow$ can we invert the exponentiation function $(g, x) \mapsto g^x$?

Common choice of $\mathbf{G}$:

- prime finite field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ (1976)
- characteristic 2 field $\mathbb{F}_{2^n}$ ($\approx$ 1979)
- elliptic curve $E(\mathbb{F}_p)$ (1985)

Discrete log problem

How fast can we invert the exponentiation function $(g, x) \mapsto g^x$?

- $g \in G$ generator, $\exists$ always a preimage $x \in \{1, \ldots, \#G\}$
- naive search, try them all: $\#G$ tests
- $O(\sqrt{\#G})$ generic algorithms

Discrete log problem

How fast can we invert the exponentiation function $(g, x) \mapsto g^x$?

- $g \in G$ generator, $\exists$ always a preimage $x \in \{1, \ldots, \#G\}$
- naive search, try them all: $\#G$ tests
- $O(\sqrt{\#G})$ generic algorithms
  - Shanks baby-step-giant-step (BSGS): $O(\sqrt{\#G})$, deterministic
  - random walk in G, cycle path finding algorithm in a connected graph (Floyd) $\rightarrow$ Pollard: $O(\sqrt{\#G})$, probabilistic
    (the cycle path encodes the answer)
  - parallel search (parallel Pollard, Kangarous)

Discrete log problem

How fast can we invert the exponentiation function $(g, x) \mapsto g^x$?

- $g \in G$ generator, $\exists$ always a preimage $x \in \{1, \ldots, \#G\}$
- naive search, try them all: $\#G$ tests
- $O(\sqrt{\#G})$ generic algorithms
    - Shanks baby-step-giant-step (BSGS): $O(\sqrt{\#G})$, deterministic
    - random walk in G, cycle path finding algorithm in a connected graph (Floyd) $\rightarrow$ Pollard: $O(\sqrt{\#G})$, probabilistic
      (the cycle path encodes the answer)
    - parallel search (parallel Pollard, Kangarous)
- independent search in each distinct subgroup
  $+$ Chinese remainder theorem (Pohlig-Hellman)

Discrete log problem

How fast can we invert the exponentiation function $(g, x) \mapsto g^x$?

$\rightarrow$ choose G of large prime order (no subgroup)

$\rightarrow$ complexity of inverting exponentiation in $O(\sqrt{\#G})$

$\rightarrow$ security level 128 bits means $\sqrt{\#G} \geq 2^{128}$
take $\#G = 2^{256}$
analogy with symmetric crypto, keylength 128 bits (16 bytes)

Discrete log problem

How fast can we invert the exponentiation function $(g, x) \mapsto g^x$?

$\rightarrow$ choose G of large prime order (no subgroup)

$\rightarrow$ complexity of inverting exponentiation in $O(\sqrt{\#G})$

$\rightarrow$ security level 128 bits means $\sqrt{\#G} \geq 2^{128}$
take $\#G = 2^{256}$
analogy with symmetric crypto, keylength 128 bits (16 bytes)

Use additional structure of G if any.

Discrete log problem when $\mathbf{G} = (\mathbb{Z}/p\mathbb{Z})^*$

Index calculus algorithm [Western–Miller 68, Adleman 79],
prequel of the Number Field Sieve algorithm (NFS)

- p prime, $(p-1)/2$ prime, $\mathbf{G} = (\mathbb{Z}/p\mathbb{Z})^*$, gen. g, target h
- get many multiplicative relations in $\mathbf{G}$
  $g^t = g_1^{e_1} g_2^{e_2} \cdots g_i^{e_i} \pmod{p}$, $g, g_1, g_2, \ldots, g_i \in \mathbf{G}$
- find a relation $h \cdot g^s = g_1^{e_1'} g_2^{e_2'} \cdots g_i^{e_i'} \pmod{p}$
- take logarithm: linear relations

$$\begin{aligned} t &= e_1 \log g_1 + e_2 \log g_2 + \ldots + e_i \log g_i \pmod{p-1} \\ &\vdots \\ \log h &= -s + e_1' \log g_1 + e_2' \log g_2 + \ldots + e_i' \log g_i \pmod{p-1} \end{aligned}$$

- solve a linear system
- get $x = \log h$

Index calculus in $(\mathbb{Z}/p\mathbb{Z})^*$: example

$p = 1109$, $r = (p-1)/4 = 277$ prime

Smoothness bound $B = 13$

$\mathcal{F}_{13} = \{2, 3, 5, 7, 11, 13\}$ small primes up to B, $i = \#\mathcal{F}$

B-smooth integer: $n = \prod_{p_i \le B} p_i^{e_i}$, p_i prime

is g^s smooth? $1 \le s \le 72$ is enough

$$\begin{array}{rcl} g^1 &=& 2 = 2 \\ g^{13} &=& 429 = 3 \cdot 11 \cdot 13 \\ g^{16} &=& 105 = 3 \cdot 5 \cdot 7 \\ g^{21} &=& 33 = 3 \cdot 11 \\ g^{44} &=& 1029 = 3 \cdot 7^3 \\ g^{72} &=& 325 = 5^2 \cdot 13 \end{array} \rightarrow \begin{array}{c} \begin{matrix} 2 & 3 & 5 & 7 & 11 & 13 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 \end{bmatrix} \end{array} \cdot \boldsymbol{x} = \begin{bmatrix} 1 \\ 13 \\ 16 \\ 21 \\ 44 \\ 72 \end{bmatrix}$$

$\boldsymbol{x} = [1, 219, 40, 34, 79, 269] \bmod 277$

$\rightarrow \log_g 7 = 34 \bmod 277$, that is, $(g^{34})^4 = 7^4$

$g^{34} = 7u$ and $u^4 = 1$

Index calculus in $(\mathbb{Z}/p\mathbb{Z})^*$: example

$\boldsymbol{x} = [1, 219, 40, 34, 79, 269] \bmod 277$

subgroup of order 4: $g_4 = g^{(p-1)/4}$

$\{1, g_4, g_4^2, g_4^3\} = \{1, 354, 1108, 755\}$

Pohlig-Hellman:

$$\begin{aligned}
3/g^{219} &= 1 = 1 \Rightarrow \log_g 3 = &&= 219 \\
5/g^{40} &= 1108 = -1 \Rightarrow \log_g 5 = 40 + (p-1)/2 &&= 594 \\
7/g^{34} &= 354 = g_4 \Rightarrow \log_g 7 = 34 + (p-1)/4 &&= 311 \\
11/g^{79} &= 755 = g_4^3 \Rightarrow \log_g 11 = 79 + 3(p-1)/4 &&= 910 \\
13/g^{269} &= 755 = g_4^3 \Rightarrow \log_g 13 = 269 + 3(p-1)/4 &&= 1100
\end{aligned}$$

$\boldsymbol{v} = [1, 219, 594, 311, 910, 1100] \bmod p-1$

Target $h = 777$

$g^{10} \cdot 777 = 495 = 3^2 \cdot 5 \cdot 11 \bmod p$

$\log_2 777 = -10 + 2\log_g 3 + \log_g 5 + \log_g 11 = 824 \bmod p-1$

$g^{824} = 777$

Index calculus in $(\mathbb{Z}/p\mathbb{Z})^*$: example

Trick

Multiplicative relations over the **integers**

$g_1, g_2, \ldots, g_i \longleftrightarrow$ small prime integers

Smooth integers $n = \prod_{p_i \leq B} p_i^{e_i}$ are quite common $\rightarrow$ it works

Complexity $e^{\sqrt{(2+o(1))(\log p)(\log \log p)}}$ (Pomerance 87)

Index calculus in $(\mathbb{Z}/p\mathbb{Z})^*$: example

Trick

Multiplicative relations over the **integers**

$g_1, g_2, \ldots, g_i \longleftrightarrow$ small prime integers

Smooth integers $n = \prod_{p_i \leq B} p_i^{e_i}$ are quite common $\rightarrow$ it works

Complexity $e^{\sqrt{(2+o(1))(\log p)(\log \log p)}}$ (Pomerance 87)

Improvements in the 80's, 90's:

- Sieve (faster relation collection)
- Smaller integers to factor
- Multiplicative relations in **number fields**
- Better **sparse linear algebra**
- Independent targets h

Sieving: Detect smooth numbers without factoring

Eratosthenes sieve

Array $T[1 \dots n-1]$ of integers from 2 up to n

At iteration i, each non-marked integer in $T[1 \dots i]$ is prime

For each non-marked $p_i = T[i]$ starting with $p_1 = T[1] = 2$:

Mark as composite all multiples $T[i + kp_i]$, $1 \leq k \leq (n-i)/p_i$

$[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]$

Sieving: Detect smooth numbers without factoring

Eratosthenes sieve

Array $T[1 \ldots n-1]$ of integers from 2 up to n
At iteration i, each non-marked integer in $T[1 \ldots i]$ is prime
For each non-marked $p_i = T[i]$ starting with $p_1 = T[1] = 2$:
    Mark as composite all multiples $T[i + kp_i]$, $1 \leq k \leq (n-i)/p_i$

$[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]$

Sieving: Detect smooth numbers without factoring

Eratosthenes sieve

Array $T[1 \ldots n-1]$ of integers from 2 up to n
At iteration i, each non-marked integer in $T[1 \ldots i]$ is prime
For each non-marked $p_i = T[i]$ starting with $p_1 = T[1] = 2$:
    Mark as composite all multiples $T[i + kp_i]$, $1 \leq k \leq (n-i)/p_i$

$[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]$

Sieving: Detect smooth numbers without factoring

Eratosthenes sieve

Array $T[1 \ldots n-1]$ of integers from 2 up to n

At iteration i, each non-marked integer in $T[1 \ldots i]$ is prime

For each non-marked $p_i = T[i]$ starting with $p_1 = T[1] = 2$:

  Mark as composite all multiples $T[i + kp_i]$, $1 \leq k \leq (n-i)/p_i$

$[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]$

Major improvement

Pomerance's Quadratic Sieve for factoring integers:
test for smoothness integers $|m| \leq A\sqrt{N}$ for some small bound A.
$\implies$ reduce the size of the integers from N to the much smaller $A\sqrt{N}$
No direct equivalent for Discrete Logarithm computation

- 1985: ElGamal, DL in GF(p^2) with two quadratic number fields, Inspired COS:
- 1986: Coppersmith–Odlyzko–Schroeppel, DL in GF(p) of complexity like the quadratic sieve

Number Field: Toy example with $\mathbb{Z}[i]$

1986: Coppersmith–Odlyzko–Schroeppel, DL in GF(p)

If $p = 1 \bmod 4$, $\exists\ U, V$ s.t. $p = U^2 + V^2$
and $|U|, |V| < \sqrt{p}$
$U/V \equiv m \mod p$ and $m^2 + 1 = 0 \mod p$

Define a map from $\mathbb{Z}[i]$ to $\mathbb{Z}/p\mathbb{Z}$

$$\begin{aligned} \phi\colon \mathbb{Z}[i] &\to \mathbb{Z}/p\mathbb{Z} \\ i &\mapsto m \bmod p \text{ where } m = U/V,\ m^2 + 1 = 0 \bmod p \end{aligned}$$

ring homomorphism $\phi(a + bi) = a + bm$

$$\phi(\underbrace{a + bi}_{\substack{\text{factor in} \\ \mathbb{Z}[i]}}) = a + bm = (a + b\ \underbrace{U/V}_{=m}) = (\underbrace{aV + bU}_{\text{factor in } \mathbb{Z}})V^{-1} \mod p$$

Example in $\mathbb{Z}[i]$

$p = 1109 = 1 \bmod 4,\ r = (p-1)/4 = 277$ prime
$p = 22^2 + 25^2$
$\max(|a|, |b|) = A = 20,\ B = 13$ smoothness bound

Example in $\mathbb{Z}[i]$

$p = 1109 = 1 \bmod 4,\ r = (p-1)/4 = 277$ prime
$p = 22^2 + 25^2$
$\max(|a|,|b|) = A = 20,\ B = 13$ smoothness bound

Rational side

$\mathcal{F}_{\text{rat}} = \{2,3,5,7,11,13\}$ primes up to B
$g(x) = Vx - U$

Example in $\mathbb{Z}[i]$

$p = 1109 = 1 \bmod 4$, $r = (p-1)/4 = 277$ prime
$p = 22^2 + 25^2$
$\max(|a|, |b|) = A = 20$, $B = 13$ smoothness bound

Rational side

$\mathcal{F}_{\text{rat}} = \{2, 3, 5, 7, 11, 13\}$ primes up to B
$g(x) = Vx - U$

Algebraic side: think about the complex number in $\mathbb{C}$

$-i(1+i)^2 = 2$, $(2+i)(2-i) = 5$, $(2+3i)(2-3i) = 13$
$\mathcal{F}_{\text{alg}} = \{1+i, 2+i, 2-i, 2+3i, 2-3i\}$
"primes" of norm up to B
$f(x) = x^2 + 1$

Example in $\mathbb{Z}[i]$

$p = 1109 = 1 \bmod 4$, $r = (p-1)/4 = 277$ prime
$p = 22^2 + 25^2$
$\max(|a|, |b|) = A = 20$, $B = 13$ smoothness bound

Rational side

$\mathcal{F}_{\text{rat}} = \{2, 3, 5, 7, 11, 13\}$ primes up to B
$g(x) = Vx - U$

Algebraic side: think about the complex number in $\mathbb{C}$

$-i(1+i)^2 = 2$, $(2+i)(2-i) = 5$, $(2+3i)(2-3i) = 13$
$\mathcal{F}_{\text{alg}} = \{1+i, 2+i, 2-i, 2+3i, 2-3i\}$
"primes" of norm up to B
$f(x) = x^2 + 1$

Units

$\mathcal{U}_{\text{alg}} = \{-1, i, -i\}$

Example in $\mathbb{Z}[i]$

$p = 1109$

$(a, b) = (-4, 7)$,
$\text{Norm}(-4 + 7i) = (-4)^2 + 7^2 = 65 = 5 \cdot 13$

In $\mathbb{Z}[i]$,

- $5 = (2 + i)(2 - i)$
- $13 = (2 + 3i)(2 - 3i)$

Example in $\mathbb{Z}[i]$

$p = 1109$

$(a, b) = (-4, 7)$,
$\text{Norm}(-4 + 7i) = (-4)^2 + 7^2 = 65 = 5 \cdot 13$

In $\mathbb{Z}[i]$,

- $5 = (2 + i)(2 - i)$
- $13 = (2 + 3i)(2 - 3i)$

$\rightarrow$ $(2 \pm i)(2 \pm 3i)$ has norm 65

$\rightarrow$ $\pm i(2 \pm i)(2 \pm 3i) = (-4 + 7i)$

We obtain $i(2 - i)(2 + 3i) = -4 + 7i$

Example in $\mathbb{Z}[i]$

$p = 1109$

$(a, b) = (-4, 7)$,
$\text{Norm}(-4 + 7i) = (-4)^2 + 7^2 = 65 = 5 \cdot 13$

In $\mathbb{Z}[i]$,

- $5 = (2 + i)(2 - i)$
- $13 = (2 + 3i)(2 - 3i)$

$\rightarrow$ $(2 \pm i)(2 \pm 3i)$ has norm 65

$\rightarrow$ $\pm i(2 \pm i)(2 \pm 3i) = (-4 + 7i)$

We obtain $i(2 - i)(2 + 3i) = -4 + 7i$

$i \leftrightarrow m = 22/25 = 755 \bmod p$
$m(2 - m)(2 + 3m) = 845 \bmod p$
$-4 + 7m = 845 \bmod p$
$(-4 \cdot 25 + 7 \cdot 22)/25 = 845 \bmod p$

Example in $\mathbb{Z}[i]$

$a+bi$	$aV+bU=$ factor in $\mathbb{Z}$	a^2+b^2	factor in $\mathbb{Z}[i]$
$-17+19i$	$-7=-7$	$650=2\cdot 5^2\cdot 13$	$i(1+i)(2+i)^2(2-3i)$
$-11+2i$	$-231=-3\cdot 7\cdot 11$	$125=5^3$	$i(2+i)^3$
$-6+17i$	$224=2^5\cdot 7$	$325=5^2\cdot 13$	$(2+i)^2(2+3i)$
$-4+7i$	$54=2\cdot 3^3$	$65=5\cdot 13$	$i(2-i)(2+3i)$
$-3+4i$	$13=13$	$25=5^2$	$-(2-i)^2$
$-2+i$	$-28=-2^2\cdot 7$	$5=5$	$-(2-i)$
$-2+3i$	$16=2^4$	$13=13$	$-(2-3i)$
$-2+11i$	$192=2^6\cdot 3$	$125=5^3$	$-(2-i)^3$
$-1+i$	$-3=-3$	$2=2$	$i(1+i)$
i	$22=2\cdot 11$	$1=1$	i
$1+3i$	$91=7\cdot 13$	$10=2\cdot 5$	$(1+i)(2+i)$
$1+5i$	$135=3^3\cdot 5$	$26=2\cdot 13$	$i(1+i)(2-3i)$
$2+i$	$72=2^3\cdot 3^2$	$5=5$	$(2+i)$
$5+i$	$147=3\cdot 7^2$	$26=2\cdot 13$	$-i(1+i)(2+3i)$

Example in $\mathbb{Z}[i]$: Matrix

Build the matrix of relations:

- one row per (a, b) pair s.t. both norms are smooth
- one column per prime of $\mathcal{F}_{\text{rat}}$
- one column for $1/V$
- one column per prime ideal of $\mathcal{F}_{\text{alg}}$
- one column per unit $(-1, i)$
- store the exponents

$$
\begin{array}{cccccccccccccc}
2 & 3 & 5 & 7 & 11 & 13 & \frac{1}{V} & -1 & i & 1+i & 2+i & 2-i & 2+3i & 2-3i
\end{array}
$$

$$
M = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 3 & 0 & 0 & 0 \\
5 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 1 & 0 \\
1 & 3 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 2 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
6 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 3 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 3 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
3 & 2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 2 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0
\end{bmatrix}
$$

$M =$

2	3	5	7	11	13	$\frac{1}{V}$	-1	i	$1+i$	$2+i$	$2-i$	$2+3i$	$2-3i$
							1	2					
			1			1	1	1	1	2			1
	1		1	1		1	1	1		3			
5			1			1				2		1	
1	3					1		1			1	1	
					1	1	1				2		
2			1			1					1		
4						1	1						1
6	1					1	1				3		
	1					1	1	1	1				
1				1		1		1					
			1		1	1			1	1			
	3	1				1		1	1				1
3	2					1				1			
	1		2			1	1	1	1			1	

$$\begin{array}{cccccccccccccc} 2 & 3 & 5 & 7 & 11 & 13 & \frac{1}{V} & -1 & i & 1+i & 2+i & 2-i & 2+3i & 2-3i \end{array}$$

$$M = \begin{bmatrix}
 & & & & & & & -1 & -2 & & & & & \\
 & & & 1 & & & 1 & -1 & -1 & -1 & -2 & & & -1 \\
 & 1 & & 1 & 1 & & 1 & -1 & -1 & & -3 & & & \\
5 & & & 1 & & & 1 & & & & -2 & & -1 & \\
1 & 3 & & & & & 1 & & -1 & & & -1 & -1 & \\
 & & & & & 1 & 1 & -1 & & & & -2 & & \\
2 & & & 1 & & & 1 & & & & & -1 & & \\
4 & & & & & & 1 & -1 & & & & & & -1 \\
6 & 1 & & & & & 1 & -1 & & & & -3 & & \\
 & 1 & & & & & 1 & -1 & -1 & -1 & & & & \\
1 & & & & 1 & & 1 & & -1 & & & & & \\
 & & & 1 & & 1 & 1 & & & -1 & -1 & & & \\
 & 3 & 1 & & & & 1 & & -1 & -1 & & & & -1 \\
3 & 2 & & & & & 1 & & & & -1 & & & \\
 & 1 & & 2 & & & 1 & -1 & -1 & -1 & & & -1 &
\end{bmatrix}$$

Example in $\mathbb{Z}[i]$

Right kernel $M \cdot \boldsymbol{x} = 0 \bmod (p-1)/4 = 277$:

$$\boldsymbol{x} = (\underbrace{1, 219, 40, 34, 79, 269}_{\text{rational side}}, \underbrace{197}_{1/V}, \underbrace{0, 0}_{\text{units}}, \underbrace{139, 84, 233, 68, 201}_{\text{algebraic side}})$$

Logarithms (in some basis)

Example in $\mathbb{Z}[i]$

Right kernel $M \cdot \boldsymbol{x} = 0 \bmod (p-1)/4 = 277$:

$$\boldsymbol{x} = (\underbrace{1, 219, 40, 34, 79, 269}_{\text{rational side}}, \underbrace{197}_{1/V}, \underbrace{0, 0}_{\text{units}}, \underbrace{139, 84, 233, 68, 201}_{\text{algebraic side}})$$

Logarithms (in some basis)

Rational side: logarithms of $\{2, 3, 5, 7, 11, 13\}$ in basis 2

$\boldsymbol{x} = [1, 219, 40, 34, 79, 269] \bmod 277$

$\rightarrow$ order 4 subgroup

$\boldsymbol{v} = [1, 219, 594, 311, 910, 1100] \bmod p - 1$

Example in $\mathbb{Z}[i]$

Right kernel $M \cdot \boldsymbol{x} = 0 \bmod (p-1)/4 = 277$:

$$\boldsymbol{x} = (\underbrace{1, 219, 40, 34, 79, 269}_{\text{rational side}}, \underbrace{197}_{1/V}, \underbrace{0, 0}_{\text{units}}, \underbrace{139, 84, 233, 68, 201}_{\text{algebraic side}})$$

Logarithms (in some basis)

Rational side: logarithms of $\{2, 3, 5, 7, 11, 13\}$ in basis 2

$\boldsymbol{x} = [1, 219, 40, 34, 79, 269] \bmod 277$

$\rightarrow$ order 4 subgroup

$\boldsymbol{v} = [1, 219, 594, 311, 910, 1100] \bmod p-1$

Target 314, generator $g = 2$

$314 = -20/7 \bmod p = -2^2 \cdot 5/7$

$$\begin{aligned} \log_g 314 &= \log_g -1 + 2\log_g 2 + \log_g 5 - \log_g 7 \\ &= (p-1)/2 + 2 + 594 - 311 = 839 \bmod p-1 \end{aligned}$$

$2^{839} = 314 \bmod p$

Number Field Sieve

Since 1993 (Gordon, Schirokauer):

$$L_p(1/3, c) = e^{(c+o(1))(\log p)^{1/3}(\log \log p)^{2/3}}$$

- polynomial selection
- **relation collection** $L_p(1/3, 1.923)$
  sieve to enumerate efficiently (a, b) pairs
- **sparse linear algebra** $L_p(1/3, 1.923)$
  compute right kernel mod prime ℓ, block-Wiedemann alg.
- individual discrete logarithm

Choosing key sizes

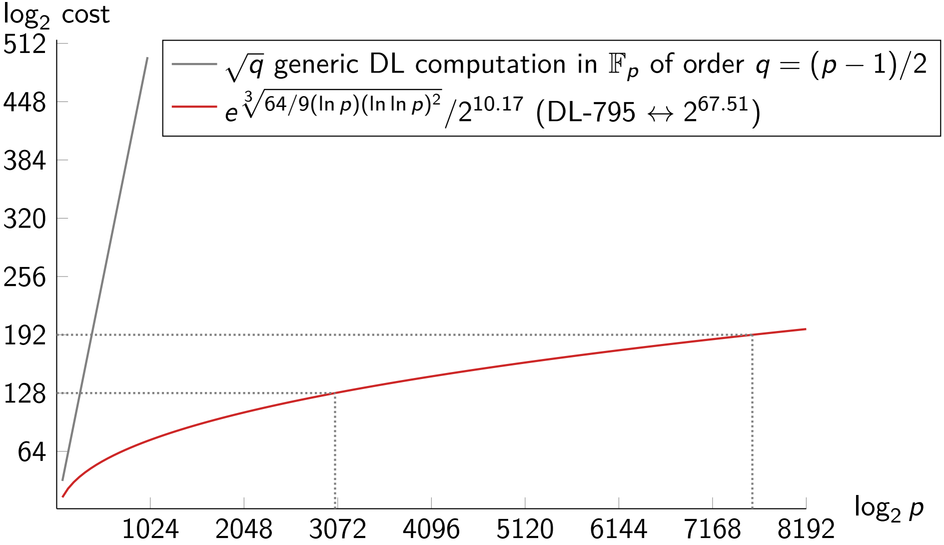
For the Discrete Log problem in $\mathbb{F}_p$ of size $\log_2(p)$ bits, $\mathbf{G}$ of order q:

n bits of security $\leftrightarrow$ the best (mathematical) attack should take at least 2^n steps

- fastest DL computation with generic algorithms: $\sqrt{q}$, q prime, divides $(p-1)/2$
- fastest DL computation in $\mathbb{F}_p$: with the Number Field Sieve algorithm
- Complexity: $\exp\left(\sqrt[3]{(64/9+o(1))(\ln p)(\ln \ln p)^2}\right)$
- $+o(1)$ not known
- $\exp\left(\sqrt[3]{(64/9+0)(\ln p_{\text{DL-795}})(\ln \ln p_{\text{DL-795}})^2}\right) = 2^{77.68}$
- DL-795 in $2^{67.51}$ operations $\rightarrow 2^{67.51}/2^{77.68} = 2^{-10.17}$

Replace unknown $+o(1)$ in the exp() by a global scaling factor $2^{-10.17} \cdot \exp()$
(A. Lenstra, Verheul, Asiacrypt'01)
This is a wrong approximation: see Le Gluher PhD thesis [LG21]

Aude Le Gluher.
Symbolic Computation and Complexity Analyses for Number Theory and Cryptography.
Phd thesis, Université de Lorraine, Nancy, France, December 2021.
https://hal.univ-lorraine.fr/tel-03564208.



DL-795: 3177 core-years, Intel Xeon Gold 6130 CPUs as a reference (2.1GHz)

$\approx 3177 \cdot 365.25 \cdot 24 \cdot 60 \cdot 60 \cdot 2.1 \cdot 10^9 \approx 2^{67.51}$

Nowadays' method: the Number Field Sieve

- developed in the 80's and 90's
- reduce the size of the numbers to be factored from $A_1\sqrt{p}$ (COS) to $A_2^d\sqrt[d]{p}$ for a smaller $A_2 < A_1$ and $d \in \{3,4,5,6\}$
- two huge steps: collecting relations, solving a large sparse system

Kevin S. McCurley.
The discrete logarithm problem.
In Carl Pomerance, editor, *Cryptology and Computational Number Theory*, volume 42 of *Proceedings of Symposia in Applied Mathematics*, pages 49–74. AMS, 1990.
https://bookstore.ams.org/psapm-42/,
http://www.mccurley.org/papers/dlog.pdf.

The development of the NFS algorithm for DL

1984 (Coppersmith: DL in small characteristic is easier, record in $\mathbb{F}_{2^{127}}$)

1985 ElGamal: Discrete logarithms in GF(p^2) with quadratic number fields

1986 Coppersmith, Odlyzko, Schroeppel:
DL computation in a prime field $\mathbb{F}_p$ with a quadratic number field (Gaussian integers)

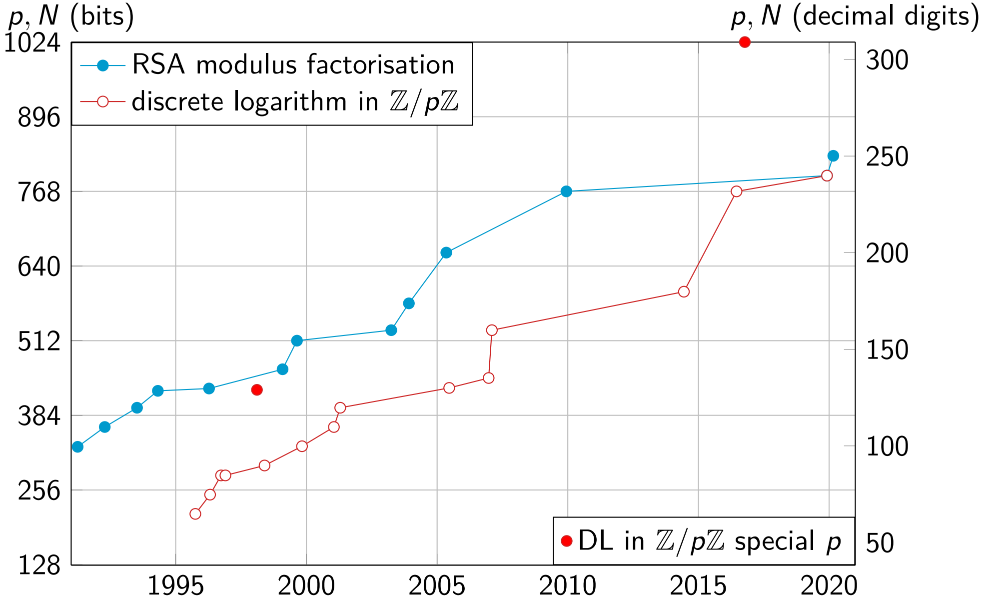
- over the period: improvements for integer factorization

1993 Gordon. Discrete Logarithms in GF(p) using the Number Field Sieve.

Arjen K. Lenstra and Hendrik W. Lenstra Jr., editors.
The development of the number field sieve, volume 1554 of *Lect. Note. Math.*
Springer, 1993.
http://doi.org/10.1007/BFb0091534

Outline

Record computations

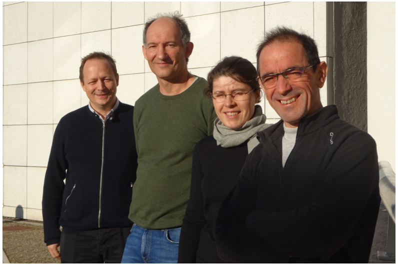


Latest record computations

Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann.
Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment.
In Daniele Micciancio and Thomas Ristenpart, eds., *CRYPTO 2020, Part II*, vol. 12171 of *LNCS*, pp. 62–91. Springer, August 2020.

Discrete logarithm computation in a 795-bit (240 dd) prime field and factorization of RSA-240 (795 bits) in December 2019, RSA-250 (829 bits) in February 2020

Video at Crypto'2020: https://youtube.com/watch?v=Qk207A4H7kU



Emmanuel, Pierrick, Aurore, Paul in Nancy.
Not on the picture: Fabrice, Nadia.

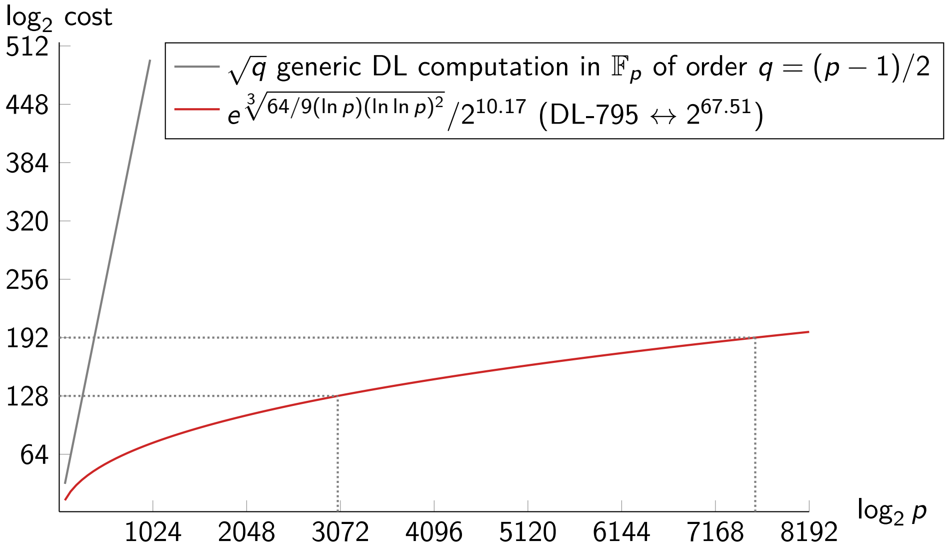
Latest record computation: DL 795 bits (240 dd)

$$
\begin{aligned}
\text{RSA-240} &= 124620366781718784065835044608106590434820374651678805754818 \\
&\quad 788883289666801188210855036039570272508747509864768438458621 \\
&\quad 054865537970253930571891217684318286362846948405301614416430 \\
&\quad 468066875699415246993185704183030512549594371372159029236099, \\
p &= \text{NextSafePrime}(N_{\text{RSA}-240}) = N_{\text{RSA}-240} + 49204 \\
q &= (p-1)/2 \text{ is prime}
\end{aligned}
$$

hardware:
Intel Xeon Gold 6130 processors, 2 CPUs, 16 physical cores/CPU, at 2.10 GHz

Discrete Logarithm 795 bits, 240 dd

$p = N + 49204$, $\ell = (p-1)/2$ prime
$f_1 = 39x^4 + 126x^3 + x^2 + 62x + 120$
$f_0 = 286512172700675411986966846394359924874576536408786368056\, x^3$
$+ 249088203007157661364751159824397355165818886038172555398 90\, x^2$
$- 1876369756001301656440395392832712103558040945994485465 2737\, x$
$- 2366104088270002562501908382208241229978789945957854322 02599$
$\text{Res}(f_0, f_1) = -540p$
More balanced integers
Smaller matrix but kernel modulo large prime ℓ



DL-795: 3177 core-years, Intel Xeon Gold 6130 CPUs as a reference (2.1GHz)

$\approx 3177 \cdot 365.25 \cdot 24 \cdot 60 \cdot 60 \cdot 2.1 \cdot 10^9 \approx 2^{67.51}$

Breaking the previous record: Why?

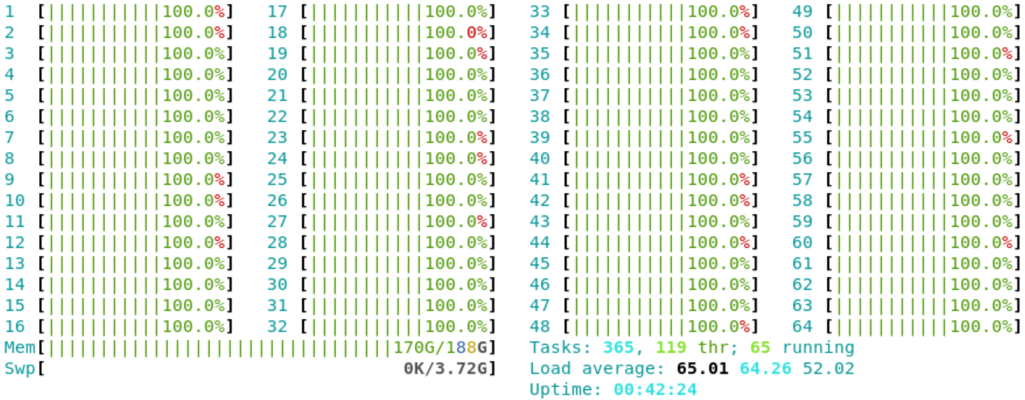
- Record computations needed for key-size recommendations
- Open-source software Cado-NFS
- Motivation to improve all the steps
- Testing folklore ideas competitive only for huge sizes (composite special-$\mathfrak{q}$, two algebraic sides)
- Exploits improvements of ECM (Bouvier–Imbert PKC'2020)
- Scaling the code for larger sizes improves the running-time on smaller sizes

The CADO-NFS software

Record computations with the CADO-NFS software.

- Important software development effort since 2007.
- 250k lines of C/C++ code, 60k for relation collection only.
- Significant improvements since 2016.
  - improved parallelism: strive to get rid of scheduling bubbles;
  - versatility: large freedom in parameter selection;
  - prediction of behaviour and yield: essential for tuning.
- Open source (LGPL), open development model (gitlab).
  Our results can be reproduced.

Relation collection looks like



Relations, matrix size, core-years timings

	RSA-240	DLP-240
polynomial selection	76 core-years	152 core-years
deg f_0, deg f_1	1, 6	3, 4
relation collection	794 core-years	2400 core-years
raw relations	8 936 812 502	3 824 340 698
unique relations	6 011 911 051	2 380 725 637
filtering	days	days
after singleton removal	$2\,603\,459\,110 \times 2\,383\,461\,671$	$1\,304\,822\,186 \times 1\,000\,258\,769$
after clique removal	$1\,175\,353\,278 \times 1\,175\,353\,118$	$149\,898\,095 \times 149\,898\,092$
after merge	282M rows, density 200	36M rows, density 253
linear algebra	83 core-years	625 core-years
characters, sqrt, ind log	days	days
total	953 core-years $\approx 2^{65.77}$ op.	3177 core-years $\approx 2^{67.51}$ op.

Intel Xeon Gold 6130 CPUs as a reference (2.1GHz)

RSA-240 and DL-795 record computations

- Parameterization strategies
- Extensive simulation framework for parameter choices
- Implementation scales well

RSA-240 and DL-795 record computations

- Parameterization strategies
- Extensive simulation framework for parameter choices
- Implementation scales well

Comparisons:

- Comparing RSA-240 to 10 years old previous record not meaningful
- Comparing DL-795 to previous record (DLP-768, 232 digits, 2016):
  On identical hardware, our DLP-795 computation would have taken
  25% less time than the 232-digits computation.
- Finite field DLP is not much harder than integer factoring.

choosing RSA modulus keysizes

- 512 bits: factorization in 7.5 h at cost \$100 on Amazon EC2
  `RSA_EXPORT` ciphersuite in SSL/TLS → FREAK attack (2015)
- 768 bits (232 dd): 2009
- 795 bits (240 dd): 2019
- 829 bits (250 dd): 2020
- 1024 bits: $\sim 2^{75}$ op. to factor, to be avoided
- 2048 bits: $\sim 2^{105}$, was standard until 2020 (ANSSI)
- 3072 bits: $\sim 2^{128}$, standard size $\iff$ 256-bit elliptic curves
- 4096 bits: $\sim 2^{145}$, high security

RSA and the quantum computer

1994: Peter Shor, algorithm for integer factorization with a quantum computer

Factorization of a n-bit integer requires a perfect quantum computer with $2n$ qbits (quantum bits)

Quantum computer extremely hard to build

Record computation in 2018: $4\ 088\ 459 = 2017 \times 2027$

RSA-1024 (bits) will be factored before a quantum compter become competitive.

Summary of RSA best practices

Use elliptic curve cryptography.
If that's not an option:

- Choose RSA modulus N at least 2048 bits, preferably 3072 bits.
- Use a good random number generator to generate primes.
- Use a secure, randomized padding scheme.

Conclusion

Slides at https://members.loria.fr/AGuillevic/teaching/

Future Milestones in the forthcoming decades: RSA-896, RSA-1024?

Outline

Attacks on discrete-logarithm based cryptosystems

- Sony Play-Station 3 (PS3) hacking, Chaos Communication Congress 2010
- Weak DH attack, 2015 https://weakdh.org/
- Weak keys in the Moscow internet voting system, 2019
  https://members.loria.fr/PGaudry/moscow/

Sony Play-Station 3 (PS3) hacking

- Revealed in 2010 at Chaos Communication Congress in Germany
- Problem of bad randomness in the ephemeral key of the ECDSA signature: Same one used to sign everything

→ With two valid signatures, the attackers can deduce Sony's private key then forge valid signatures themselves for anything

ECDSA signature, NIST FIPS 186-4, updated to 186-5 (February 3, 2023)

Domain parameters

- field size $q = p$ an odd prime or $q = 2^m$ a binary field
- elliptic curve parameters: curve type (Koblitz, binary, short Weierstrass, Montgomery), curve coefficients a, b,
- group $\mathbf{G}$ parameters: prime order $n = \#\mathbf{G}$, curve cofactor h, $G = (x_G, y_G)$ a generator of order n, optional domain parameter seed

Key pair (d, P) generation, secret d and public P

- generate a private secret random $0 < d < n$ (in the scalar field)
- compute the public key: curve point $P = [d]G$

ECDSA signature of a message m, under the private key d

- generate a new secret random ephemeral key $k \leftarrow \{1, \ldots, n-1\}$
- compute its inverse $k^{-1} \bmod n$
- compute $R = [k]G = (x_R, y_R)$ and set $r = x_R$
- compute the signature (r, s) with

$$s = k^{-1} \cdot (H(m) + r \cdot d) \bmod n$$

- securely erase k and k^{-1}

Moreover the standard specifies how to generate random ephemeral keys k_i and how to select a secure cryptographic hash function H.

ECDSA signature of a message m, under the private key d

- generate a new secret random ephemeral key $k \leftarrow \{1, \ldots, n-1\}$
- compute its inverse $k^{-1} \bmod n$
- compute $R = [k]G = (x_R, y_R)$ and set $r = x_R$
- compute the signature (r, s) with

$$s = k^{-1} \cdot (H(m) + r \cdot d) \bmod n$$

- securely erase k and k^{-1}

Moreover the standard specifies how to generate random ephemeral keys k_i and how to select a secure cryptographic hash function H.

Verify (r, s): with $P = [d]G$, check that Q has $x_Q = r \bmod n$, with

$$\begin{aligned} Q &= [s^{-1} \cdot H(m) \bmod n]G + [s^{-1} \cdot r \bmod n]P = (x_Q, y_Q) \\ &= [s^{-1}(H(m) + r \cdot d)]G \stackrel{?}{=} R = [k]G \end{aligned}$$

PS3 attack (2010)

Same ephemeral key k used to sign different messages, say m_1, m_2

- $(r, s_1 = k^{-1} \cdot (H(m_1) + r \cdot d) \bmod n)$
- $(r, s_2 = k^{-1} \cdot (H(m_2) + r \cdot d) \bmod n)$

Recover the private key d

- compute the difference $s_1 - s_2 = k^{-1} \cdot (H(m_1) - H(m_2)) \bmod n$
- the secret part $r \cdot d$ vanished!
- publicly compute $H(m_1) - H(m_2) \bmod n$ and recover the ephemeral secret key

$$k = (s_1 - s_2)^{-1} \cdot (H(m_1) - H(m_2)) \bmod n$$

- from (r, s_1) and k, recover $d = (k \cdot s_1 - H(m_1)) \cdot r^{-1} \bmod n$

Knowing the manufacturer's private key d allows anyone to sign any non-legitimate documents (software, games for the PS3). The signature will be accepted as valid by any verifier.

Weak Diffie–Hellman and the Logjam attack (2015)

https://weakdh.org/

- inspired by the FREAK attack
- Active TLS MITM (Malicious Intruder in The Middle) downgrade attack
- Force use of `DHE_EXPORT` cipher suite (Diffie–Hellman Ephemeral key-exchange) with 512-bit prime p
- precomputation of a huge database of the discrete logarithms of a factor basis so as to get a targeted individual discrete log in live

TLS 1.2 Handshake reference and tutorial:
https://datatracker.ietf.org/doc/html/rfc5246
https://tlseminar.github.io/first-few-milliseconds/

What makes the attack possible?

MITM

- No signature of the cipher suite chosen (`DHE` vs `DHE_EXPORT`)
- The attacker can intercept the communication
- the attacker convinces the server that the browser wants `DHE_EXPORT`
- the attacker answers back `DHE` and fools the browser with the server's `DHE_EXPORT` 512-bit prime p
- the attack works because the browser does not check the keysizes (512 bits) of the server's parameters
- real-time discrete log computation to hack the MACs in the Finished messages

Regular Diffie–Hellman Ephemeral key-exchange in TLS 1.2

Client **Server**

ClientHello $=$ {supported cipher suites}, client random nonce cr →

Regular Diffie–Hellman Ephemeral key-exchange in TLS 1.2

Client **Server**

ClientHello = {supported cipher suites}, client random nonce *cr* →

← ServerHello = chosen cipher suite, server random nonce *sr*

Regular Diffie–Hellman Ephemeral key-exchange in TLS 1.2

Client **Server**

ClientHello = {supported cipher suites}, client random nonce *cr* →

← ServerHello = chosen cipher suite, server random nonce *sr*

← Certificate = Server public RSA key *S*, CA signatures chain

Regular Diffie–Hellman Ephemeral key-exchange in TLS 1.2

Client **Server**

ClientHello = {supported cipher suites}, client random nonce cr

ServerHello = chosen cipher suite, server random nonce sr

Certificate = Server public RSA key S, CA signatures chain

ServerKeyExchange = p, g, g^b, $\text{Sign}_{\text{RSA key } S}(cr, sr, p, g, g^b)$

Regular Diffie–Hellman Ephemeral key-exchange in TLS 1.2

Client → **Server**

Client → Server: ClientHello $=$ {supported cipher suites}, client random nonce cr

Server → Client: ServerHello $=$ chosen cipher suite, server random nonce sr

Server → Client: Certificate $=$ Server public RSA key S, CA signatures chain

Server → Client: ServerKeyExchange $= p, g, g^b$, $\mathrm{Sign}_{\text{RSA key } S}(cr, sr, p, g, g^b)$

Client → Server: ClientKeyExchange $= g^a$

Regular Diffie–Hellman Ephemeral key-exchange in TLS 1.2

Client — **Server**

Client → Server: ClientHello = {supported cipher suites}, client random nonce cr

Server → Client: ServerHello = chosen cipher suite, server random nonce sr

Server → Client: Certificate = Server public RSA key S, CA signatures chain

Server → Client: ServerKeyExchange = p, g, g^b, $\text{Sign}_{\text{RSA key } S}(cr, sr, p, g, g^b)$

Client → Server: ClientKeyExchange = g^a

Client: $\text{kdf}(g^{ab}, cr, sr) \rightarrow k_{m_c}, k_{m_s}, k_e$

g^{ab} = pre-master secret

key **d**erivation **f**unction

Server: $\text{kdf}(g^{ab}, cr, sr) \rightarrow k_{m_c}, k_{m_s}, k_e$

Regular Diffie–Hellman Ephemeral key-exchange in TLS 1.2

Client **Server**

ClientHello = {supported cipher suites}, client random nonce cr →

← ServerHello = chosen cipher suite, server random nonce sr

← Certificate = Server public RSA key S, CA signatures chain

← ServerKeyExchange = p, g, g^b, $\text{Sign}_{\text{RSA key } S}(cr, sr, p, g, g^b)$

ClientKeyExchange = g^a →

$\text{kdf}(g^{ab}, cr, sr)$ g^{ab} = pre-master secret $\text{kdf}(g^{ab}, cr, sr)$

$\rightarrow k_{m_c}, k_{m_s}, k_e$ **k**ey **d**erivation **f**unction $\rightarrow k_{m_c}, k_{m_s}, k_e$

MAC = **M**essage **A**uthentication **C**ode

ClientFinished = $\text{MAC}_{k_{m_c}}$(Client's view of handshake) →

Regular Diffie–Hellman Ephemeral key-exchange in TLS 1.2

Client **Server**

ClientHello = {supported cipher suites}, client random nonce cr →

← ServerHello = chosen cipher suite, server random nonce sr

← Certificate = Server public RSA key S, CA signatures chain

← ServerKeyExchange = p, g, g^b, $\text{Sign}_{\text{RSA key } S}(cr, sr, p, g, g^b)$

ClientKeyExchange = g^a →

kdf(g^{ab}, cr, sr) $\to k_{m_c}, k_{m_s}, k_e$

g^{ab} = pre-master secret

key **d**erivation **f**unction

kdf(g^{ab}, cr, sr) $\to k_{m_c}, k_{m_s}, k_e$

MAC = **M**essage **A**uthentication **C**ode

ClientFinished = $\text{MAC}_{k_{m_c}}$(Client's view of handshake) →

← ServerFinished = $\text{MAC}_{k_{m_c}}$(Server's view of handshake)

Regular Diffie–Hellman Ephemeral key-exchange in TLS 1.2

Client **Server**

ClientHello = {supported cipher suites}, client random nonce cr →

← ServerHello = chosen cipher suite, server random nonce sr

← Certificate = Server public RSA key S, CA signatures chain

← ServerKeyExchange = p, g, g^b, $\text{Sign}_{\text{RSA key } S}(cr, sr, p, g, g^b)$

ClientKeyExchange = g^a →

$\text{kdf}(g^{ab}, cr, sr) \to k_{m_c}, k_{m_s}, k_e$

g^{ab} = pre-master secret

key **d**erivation **f**unction

$\text{kdf}(g^{ab}, cr, sr) \to k_{m_c}, k_{m_s}, k_e$

MAC = **M**essage **A**uthentication **C**ode

ClientFinished = $\text{MAC}_{k_{m_c}}$(Client's view of handshake) →

← ServerFinished = $\text{MAC}_{k_{m_c}}$(Server's view of handshake)

Verify Server's MAC

Verify Client's MAC

MITM DHE attack

Client **Attacker** **Server**

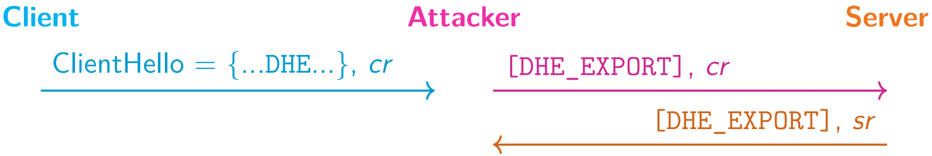
MITM DHE attack

Client **Attacker** **Server**

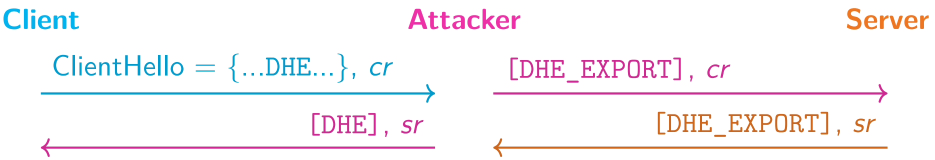
$\text{ClientHello} = \{\ldots\texttt{DHE}\ldots\}, \ cr$ →

MITM DHE attack

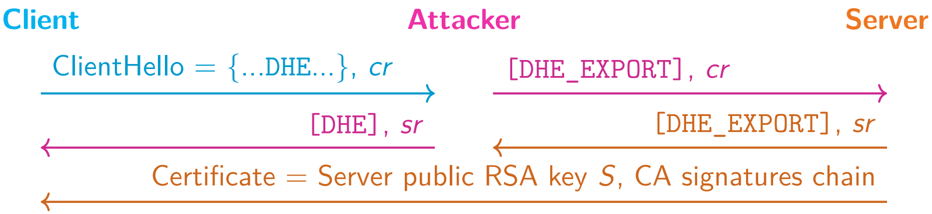
MITM DHE attack



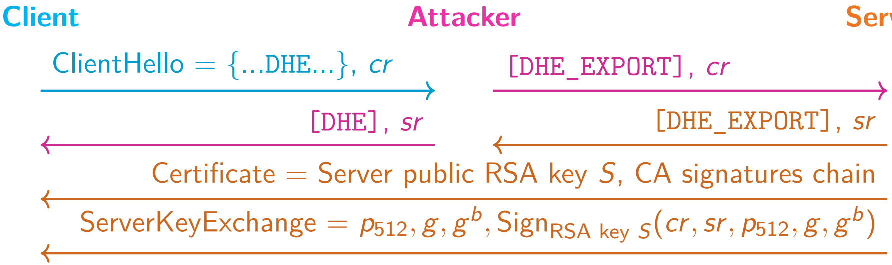
MITM DHE attack



MITM DHE attack



MITM DHE attack



MITM DHE attack

Client **Attacker** **Server**

ClientHello = {...DHE...}, *cr* (Client → Attacker)

[DHE_EXPORT], *cr* (Attacker → Server)

[DHE], *sr* (Attacker → Client)

[DHE_EXPORT], *sr* (Server → Attacker)

Certificate = Server public RSA key *S*, CA signatures chain (Server → Client)

ServerKeyExchange = $p_{512}, g, g^b, \mathrm{Sign}_{\text{RSA key } S}(cr, sr, p_{512}, g, g^b)$ (Server → Client)

length(p_{512}) not checked

MITM DHE attack

Client **Attacker** **Server**

ClientHello = {...DHE...}, *cr* (Client → Attacker)

[DHE_EXPORT], *cr* (Attacker → Server)

[DHE], *sr* (Attacker → Client)

[DHE_EXPORT], *sr* (Server → Attacker)

Certificate = Server public RSA key S, CA signatures chain

ServerKeyExchange = $p_{512}, g, g^b, \text{Sign}_{\text{RSA key } S}(cr, sr, p_{512}, g, g^b)$

length(p_{512}) not checked

ClientKeyExchange = g^a (Client → Attacker)

MITM DHE attack

Client **Attacker** **Server**

ClientHello $= \{\ldots\texttt{DHE}\ldots\}$, cr $\longrightarrow$

$\texttt{[DHE\_EXPORT]}$, cr $\longrightarrow$

$\longleftarrow$ $\texttt{[DHE]}$, sr

$\longleftarrow$ $\texttt{[DHE\_EXPORT]}$, sr

$\longleftarrow$ Certificate $=$ Server public RSA key S, CA signatures chain

$\longleftarrow$ ServerKeyExchange $= p_{512}, g, g^b, \mathsf{Sign}_{\text{RSA key } S}(cr, sr, p_{512}, g, g^b)$

length(p_{512}) not checked

ClientKeyExchange $= g^a$ $\longrightarrow$

kdf(g^{ab}, cr, sr)

$\rightarrow k_{m_c}, k_{m_s}, k_e$

MITM DHE attack

Client **Attacker** **Server**

ClientHello = {...DHE...}, cr →

[DHE_EXPORT], cr →

← [DHE], sr

← [DHE_EXPORT], sr

← Certificate = Server public RSA key S, CA signatures chain

← ServerKeyExchange = $p_{512}, g, g^b, \mathrm{Sign}_{\mathrm{RSA\ key}\ S}(cr, sr, p_{512}, g, g^b)$

length(p_{512}) not checked

ClientKeyExchange = g^a →

kdf(g^{ab}, cr, sr)

$\rightarrow k_{m_c}, k_{m_s}, k_e$

$\mathrm{MAC}_{k_{m_c}}$(DHE...) →

MITM DHE attack

Client **Attacker** **Server**

ClientHello = {...DHE...}, cr → (Client → Attacker)

[DHE_EXPORT], cr → (Attacker → Server)

← [DHE], sr (Attacker → Client)

← [DHE_EXPORT], sr (Server → Attacker)

← Certificate = Server public RSA key S, CA signatures chain

← ServerKeyExchange = p_{512}, g, g^b, $\text{Sign}_{\text{RSA key } S}(cr, sr, p_{512}, g, g^b)$

length(p_{512}) not checked

ClientKeyExchange = g^a →

kdf(g^{ab}, cr, sr)

$\rightarrow k_{m_c}, k_{m_s}, k_e$

$\text{MAC}_{k_{m_c}}$(DHE...) →

How to hack the Server's $\text{MAC}_{k_{m_c}}$?

MITM DHE attack

Client **Attacker** **Server**

ClientHello $= \{\ldots\texttt{DHE}\ldots\}, cr$ →

[DHE_EXPORT], cr →

← [DHE], sr

← [DHE_EXPORT], sr

← Certificate $=$ Server public RSA key S, CA signatures chain

← ServerKeyExchange $= p_{512}, g, g^b, \mathsf{Sign}_{\text{RSA key } S}(cr, sr, p_{512}, g, g^b)$

length(p_{512}) not checked

ClientKeyExchange $= g^a$ →

$\mathrm{kdf}(g^{ab}, cr, sr)$

$\rightarrow k_{m_c}, k_{m_s}, k_e$

$\mathrm{MAC}_{k_{m_c}}(\texttt{DHE}\ldots)$ →

How to hack the Server's $\mathbf{MAC}_{k_{m_c}}$?

Keep Client waiting for ServerFinished

MITM DHE attack

Client **Attacker** **Server**

Client → Attacker: ClientHello = {...DHE...}, cr

Attacker → Server: [DHE_EXPORT], cr

Attacker → Client: [DHE], sr

Server → Attacker: [DHE_EXPORT], sr

Server → Client: Certificate = Server public RSA key S, CA signatures chain

Server → Client: ServerKeyExchange = p_{512}, g, g^b, $\text{Sign}_{\text{RSA key } S}(cr, sr, p_{512}, g, g^b)$

length(p_{512}) not checked

Client → Attacker: ClientKeyExchange = g^a

kdf(g^{ab}, cr, sr)
$\rightarrow k_{m_c}, k_{m_s}, k_e$

Client → Attacker: $\text{MAC}_{k_{m_c}}$(DHE...)

How to hack the Server's $\text{MAC}_{k_{m_c}}$?

Keep Client waiting for ServerFinished

Solve $\log_g(g^b) = b$, get g^{ab},

kdf(g^{ab}, cr, sr) $\rightarrow k_{m_c}, k_{m_s}, k_e$

MITM DHE attack

Client — **Attacker** — **Server**

Client → Attacker: ClientHello $= \{\ldots\texttt{DHE}\ldots\}, cr$

Attacker → Server: $\texttt{[DHE\_EXPORT]}, cr$

Server → Attacker: $\texttt{[DHE\_EXPORT]}, sr$

Attacker → Client: $\texttt{[DHE]}, sr$

Server → Client: Certificate $=$ Server public RSA key S, CA signatures chain

Server → Client: ServerKeyExchange $= p_{512}, g, g^b, \mathrm{Sign}_{\mathrm{RSA\ key\ }S}(cr, sr, p_{512}, g, g^b)$

length(p_{512}) not checked

Client → Attacker: ClientKeyExchange $= g^a$

$\mathrm{kdf}(g^{ab}, cr, sr)$

$\rightarrow k_{m_c}, k_{m_s}, k_e$

Client → Attacker: $\mathrm{MAC}_{k_{m_c}}(\texttt{DHE}\ldots)$

Attacker → Client: $\mathrm{MAC}_{k_{m_c}}(\texttt{DHE}\ldots)$

How to hack the Server's $\mathbf{MAC}_{k_{m_c}}$?

Keep Client waiting for ServerFinished

Solve $\log_g(g^b) = b$, get g^{ab},

$\mathrm{kdf}(g^{ab}, cr, sr) \rightarrow k_{m_c}, k_{m_s}, k_e$

MITM DHE attack

Client **Attacker** **Server**

Client → Attacker: ClientHello $= \{\ldots\texttt{DHE}\ldots\}, cr$

Attacker → Server: $\texttt{[DHE\_EXPORT]}, cr$

Attacker → Client: $\texttt{[DHE]}, sr$

Server → Attacker: $\texttt{[DHE\_EXPORT]}, sr$

Server → Client: Certificate = Server public RSA key S, CA signatures chain

Server → Client: ServerKeyExchange $= p_{512}, g, g^b, \text{Sign}_{\text{RSA key } S}(cr, sr, p_{512}, g, g^b)$

length(p_{512}) not checked

Client → Attacker: ClientKeyExchange $= g^a$

$\text{kdf}(g^{ab}, cr, sr)$
$\rightarrow k_{m_c}, k_{m_s}, k_e$

Client → Attacker: $\text{MAC}_{k_{m_c}}(\texttt{DHE}\ldots)$

Attacker → Client: $\text{MAC}_{k_{m_c}}(\texttt{DHE}\ldots)$

How to hack the Server's $\text{MAC}_{k_{m_c}}$?

Keep Client waiting for ServerFinished

Solve $\log_g(g^b) = b$, get g^{ab},

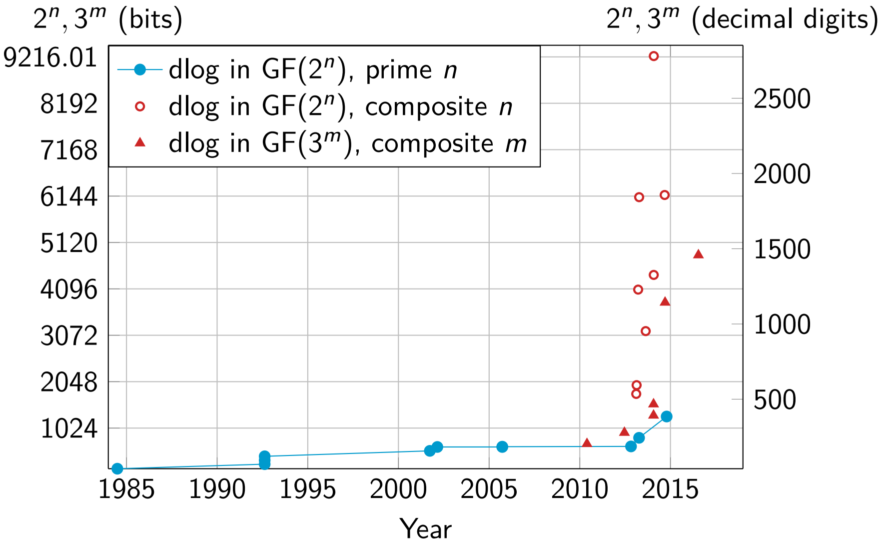
$\text{kdf}(g^{ab}, cr, sr) \rightarrow k_{m_c}, k_{m_s}, k_e$

Attacker impersonated the Server

Weak keys in the Moscow internet voting system (2019)

https://members.loria.fr/PGaudry/moscow/

Discrete logarithm computation in finite fields $\mathbb{F}_{2^n}$ and $\mathbb{F}_{3^m}$



Outline

What is a pairing?

$(\mathbf{G}_1, +), (\mathbf{G}_2, +), (\mathbf{G}_3, \cdot)$ three cyclic groups of order r
Pairing: map $e : \mathbf{G}_1 \times \mathbf{G}_2 \to \mathbf{G}_3$

1. bilinear: $e(P_1 + P_2, Q) = e(P_1, Q) \cdot e(P_2, Q)$, $e(P, Q_1 + Q_2) = e(P, Q_1) \cdot e(P, Q_2)$
2. non-degenerate: $e(G_1, G_2) \neq 1$ for $\langle G_1 \rangle = \mathbf{G}_1$, $\langle G_2 \rangle = \mathbf{G}_2$
3. efficiently computable.

In practice we use mostly

$$e([a]P, [b]Q) = e([b]P, [a]Q) = e(P, Q)^{ab} \ .$$

$\rightsquigarrow$ Many applications in asymmetric cryptography.

Pairings in cryptography: 1993 and 2001

1993

Menezes–Okamoto–Vanstone attack

2001

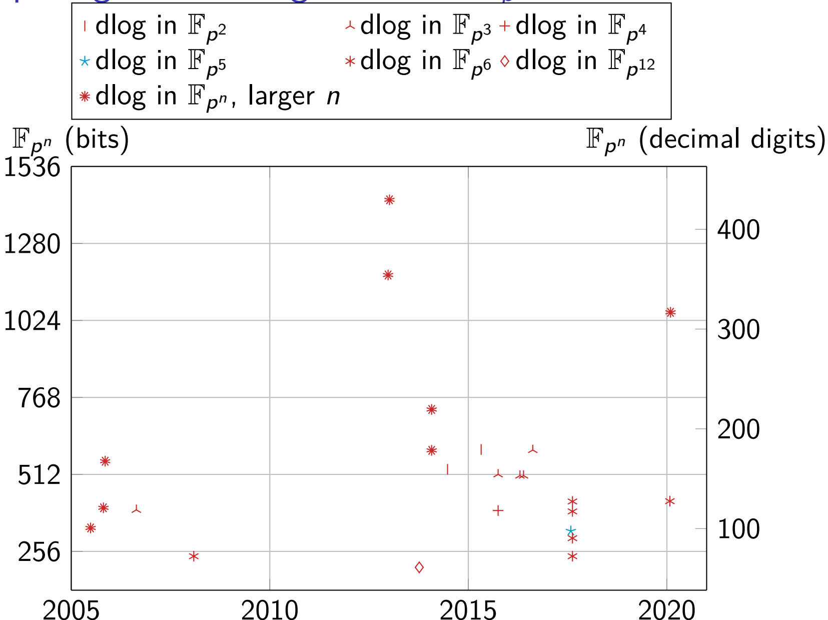
- Joux' tri-partite key exchange
- Boneh Frankin Identity based encryption
- Boneh Lynn Shacham short signature

Pairings with curves over fields $\mathbb{F}_{2^n}$ and $\mathbb{F}_{3^m}$, rise and fall

Pairings with curves over fields $\mathbb{F}_p$

https://members.loria.fr/AGuillevic/pairing-friendly-curves/

Computing Discrete logarithms in $\mathbb{F}_{p^n}$



Choosing key-sizes

https://members.loria.fr/AGuillevic/pairing-friendly-curves/