Pre-quantum and post-quantum variants of the Diffie-Hellman protocol

Francisco Rodríguez-Henríquez
Cinvestav, México

7th International Conference on Mathematics and Computing (ICMC 2021)
Shibpur, India, March 5, 2021

Main primitives and building blocks in modern cryptography

Main primitives and building blocks in modern cryptography

- Primitives:
  - Encryption/decryption of digital documents [this task is typically solved using symmetric cryptography]

Main primitives and building blocks in modern cryptography

- Primitives:
  - Encryption/decryption of digital documents [this task is typically solved using symmetric cryptography]
  - Signature/verification of digital documents [This task is usually solved using public key cryptography]

Main primitives and building blocks in modern cryptography

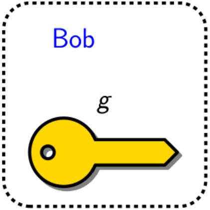
- Primitives:
  - Encryption/decryption of digital documents [this task is typically solved using symmetric cryptography]
  - Signature/verification of digital documents [This task is usually solved using public key cryptography]
  - Sharing a secret among two or more parties [this task is usually solved using the Diffie-Hellman protocol or its variants]

Main primitives and building blocks in modern cryptography

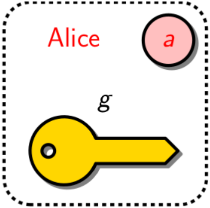
- Primitives:
  - Encryption/decryption of digital documents [this task is typically solved using symmetric cryptography]
  - Signature/verification of digital documents [This task is usually solved using public key cryptography]
  - Sharing a secret among two or more parties [this task is usually solved using the Diffie-Hellman protocol or its variants]
- Building blocks:
  - Block ciphers and stream ciphers
  - Hash functions
  - Public key crypto-schemes
  - ...

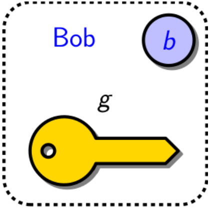
Design problem: How to share a secret?

Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976

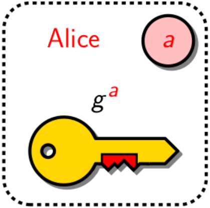


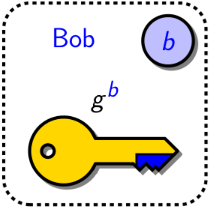
Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976



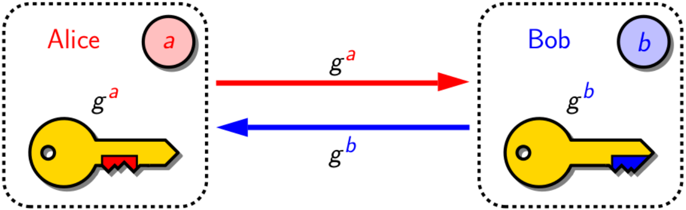


Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976

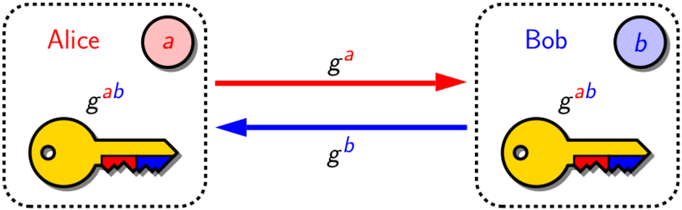




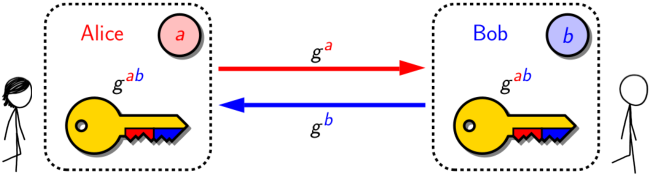
Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976



Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976



Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976

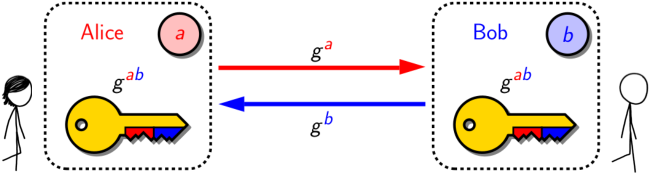


- Alice and Bob decide to work in the $\mathbb{Z}_p$ group, with p a large odd prime. They also choose a generator $g \in \mathbb{Z}_p$ (i.e., $Ord(g) = p - 1$).
- Alice and Bob select $a, b \in \mathbb{Z}_p$, respectively
- Alice and Bob compute a shared secret as,

$$K = (g^a)^b = (g^b)^a$$

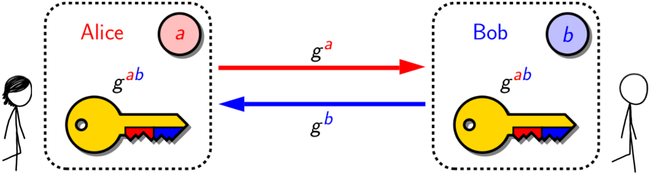
Note: This protocol can only be secure against passive attackers

Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976



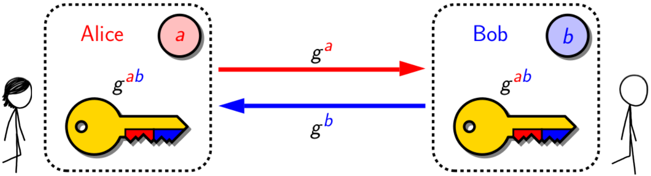
Protocol's security lies in the computational intractability of solving the Discrete Logarithm Problem (DLP), namely,
Given a prime p and a generator $g, h \in [1, p-1]$, find an integer k such that, $g^k \equiv h \bmod p$.

Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976



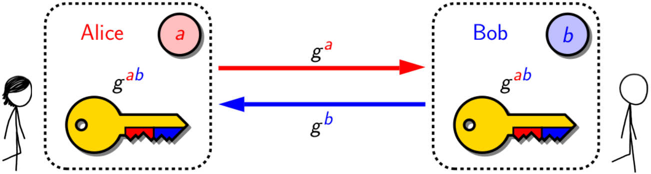
Protocol's security lies in the computational intractability of solving the Discrete Logarithm Problem (DLP), namely,
Given a prime p and a generator $g, h \in [1, p-1]$, find an integer k such that, $g^k \equiv h \bmod p$.

Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976

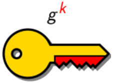


Protocol's security lies in the computational intractability of solving the Discrete Logarithm Problem (DLP), namely,
Given a prime p and a generator $g, h \in [1, p-1]$, find an integer k such that, $g^k \equiv h \bmod p$.

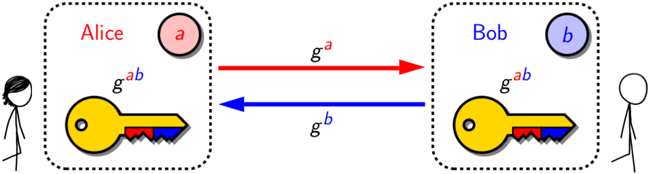
Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976



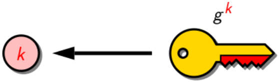
Protocol's security lies in the computational intractability of solving the Discrete Logarithm Problem (DLP), namely,
Given a prime p and a generator g, $h \in [1, p-1]$, find an integer k such that, $g^k \equiv h \bmod p$.



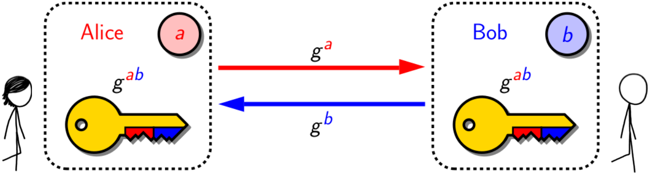
Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976



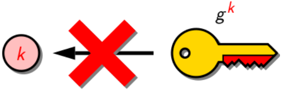
Protocol's security lies in the computational intractability of solving the Discrete Logarithm Problem (DLP), namely,
Given a prime p and a generator $g, h \in [1, p-1]$, find an integer k such that, $g^k \equiv h \bmod p$.



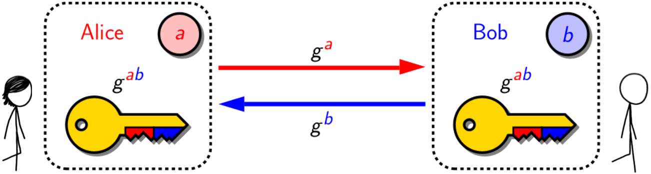
Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976



Protocol's security lies in the computational intractability of solving the Discrete Logarithm Problem (DLP), namely,
Given a prime p and a generator g, $h \in [1, p-1]$, find an integer k such that,
$g^k \equiv h \bmod p$.

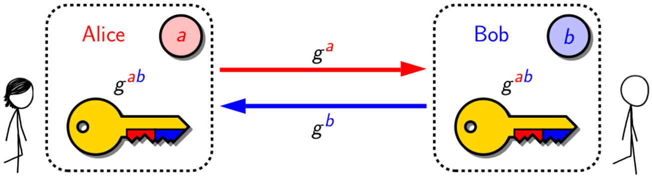


Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976



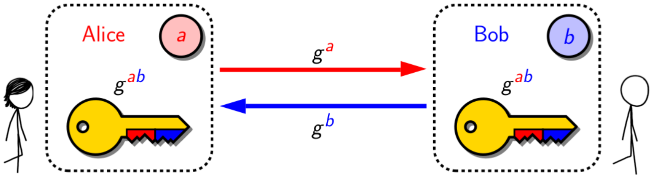
- Diffie and Hellman published their protocol in their breakthrough paper, Diffie, W.; Hellman, M. (1976). "New directions in cryptography". IEEE Transactions on Information Theory. 22 (6): 644–654.

Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976



- Diffie and Hellman published their protocol in their breakthrough paper, Diffie, W.; Hellman, M. (1976). "New directions in cryptography". IEEE Transactions on Information Theory. 22 (6): 644–654.
  "We stand today on the brink of a revolution in cryptography"

Design problem: How to share a secret?. Solution: Diffie-Hellman Protocol 1976



- Diffie and Hellman published their protocol in their breakthrough paper, Diffie, W.; Hellman, M. (1976). "New directions in cryptography". IEEE Transactions on Information Theory. 22 (6): 644–654.
  "We stand today on the brink of a revolution in cryptography"
- Diffie and Hellman won the 2015 Turing award
- Since its publication in 1976, "New directions in cryptography" has inspired many new ideas in the discipline.
  In this talk we will revisit four different versions of this protocol [!!]

Hard computational problems

1. Integer factorization problem: Given an integer $N = p \cdot q$ find its prime factors p and q. Find p, q such that $2019 = p \cdot q$

Hard computational problems

1. Integer factorization problem: Given an integer $N = p \cdot q$ find its prime factors p and q. Find p, q such that $2019 = p \cdot q$
   answer: $2021 = 43 \cdot 47$

Hard computational problems

1. Integer factorization problem: Given an integer $N = p \cdot q$ find its prime factors p and q. Find p, q such that $2019 = p \cdot q$
   answer: $2021 = 43 \cdot 47$
2. Discrete logarithm problem: Given a prime p and $g, h \in [1, p-1]$, find an integer x (if one exists) such that, $g^x \equiv h \bmod p$.
   find x such that $2^x \equiv 304 \bmod 419$

Hard computational problems

1. Integer factorization problem: Given an integer $N = p \cdot q$ find its prime factors p and q. Find p, q such that $2019 = p \cdot q$
   answer: $2021 = 43 \cdot 47$
2. Discrete logarithm problem: Given a prime p and $g, h \in [1, p-1]$, find an integer x (if one exists) such that, $g^x \equiv h \bmod p$.
   find x such that $2^x \equiv 304 \bmod 419$
   answer: $2^{343} \equiv 304 \bmod 419$.

Hard computational problems

1. Integer factorization problem: Given an integer $N = p \cdot q$ find its prime factors p and q. Find p, q such that $2019 = p \cdot q$
   answer: $2021 = 43 \cdot 47$
2. Discrete logarithm problem: Given a prime p and $g, h \in [1, p-1]$, find an integer x (if one exists) such that, $g^x \equiv h \bmod p$.
   find x such that $2^x \equiv 304 \bmod 419$
   answer: $2^{343} \equiv 304 \bmod 419$.
   More generally: Given $g, h \in \mathbb{F}_q^*$, find an integer x (if one exists) such that, $g^x \equiv h$, where $q = p^k$ is the power of a prime

Hard computational problems

1. Integer factorization problem: Given an integer $N = p \cdot q$ find its prime factors p and q. Find p, q such that $2019 = p \cdot q$
   answer: $2021 = 43 \cdot 47$
2. Discrete logarithm problem: Given a prime p and $g, h \in [1, p-1]$, find an integer x (if one exists) such that, $g^x \equiv h \bmod p$.
   find x such that $2^x \equiv 304 \bmod 419$
   answer: $2^{343} \equiv 304 \bmod 419$.
   More generally: Given $g, h \in \mathbb{F}_q^*$, find an integer x (if one exists) such that, $g^x \equiv h$, where $q = p^k$ is the power of a prime
3. Elliptic curve discrete logarithm problem: Given an elliptic curve $E/\mathbb{F}_q$ and $P, Q \in E(\mathbb{F}_q)$, find an integer x (if one exists) such that, $xP = Q$ [More ECDLP material will be discussed later]

Running time complexity

- The efficiency of an algorithm is measured in terms of its input size.

Running time complexity

- The efficiency of an algorithm is measured in terms of its input size.
  - For the discrete logarithm problem in $\mathbb{F}_q$, the input size is $O(\log q)$ bits.

Running time complexity

- The efficiency of an algorithm is measured in terms of its input size.
  - For the discrete logarithm problem in $\mathbb{F}_q$, the input size is $O(\log q)$ bits.
- A polynomial-time algorithm is one whose running time is bounded by a polynomial in the input size: $(\log q)^c$, where c is a constant.

Running time complexity

- The efficiency of an algorithm is measured in terms of its input size.
  - For the discrete logarithm problem in $\mathbb{F}_q$, the input size is $O(\log q)$ bits.
- A polynomial-time algorithm is one whose running time is bounded by a polynomial in the input size: $(\log q)^c$, where c is a constant.
- A fully exponential-time algorithm is one whose running time is of the form q^c, where c is a constant.

Running time complexity

- The efficiency of an algorithm is measured in terms of its input size.
  - For the discrete logarithm problem in $\mathbb{F}_q$, the input size is $O(\log q)$ bits.
- A polynomial-time algorithm is one whose running time is bounded by a polynomial in the input size: $(\log q)^c$, where c is a constant.
- A fully exponential-time algorithm is one whose running time is of the form q^c, where c is a constant.
- A subexponential-time algorithm as one whose running time is of the form,

$$L_q[\alpha, c] = e^{c(\log q)^{\alpha}(\log \log q)^{1-\alpha}},$$

where $0 < \alpha < 1$, and c is a constant.

$\alpha = 0$: polynomial $\alpha = 1$: fully exponential

Attacks on discrete log computation over small char $\mathbb{F}_{q^n}$: Main developments in the last 30+ years

Let Q be defined as $Q = q^n$.

- Hellman-Reyneri 1982: Index-calculus $L_Q[\frac{1}{2}, 1.414]$
- Coppersmith 1984: $L_Q[\frac{1}{3}, 1.526]$
- Joux-Lercier 2006: $L_Q[\frac{1}{3}, 1.442]$ when q and n are "balanced"
- Hayashi et al. 2012: Used an improved version of the Joux-Lercier method to compute discrete logs over the field $\mathbb{F}_{3^{6\cdot 97}}$
- Joux 2012: $L_Q[\frac{1}{3}, 0.961]$ when q and n are "balanced"
- Joux 2013: $L_Q[\frac{1}{4} + o(1), c]$ when $Q = q^{d\cdot m}$, d a small integer (e.g. $d = 2, 3$) and $q \approx m$
- Göloğlu et al. 2013: similar to Joux 2013, BPA @ Crypto'2013

Attacks on discrete log computation over small char $\mathbb{F}_{q^{3n}}$: security level consequences

Let us assume that one wants to compute discrete logarithms in the field $\mathbb{F}_{q^{3n}}$, with $q = 3^6$, $n = 509$, Notice that the group size of that field is,

$$\#\mathbb{F}_{3^{6\cdot 509}} = \lceil \log_2(3) \cdot 6 \cdot 509 \rceil = 4841 \text{ bits.}$$

Algorithm	Time complexity	Equiv. bit security level
Hellman-Reyneri 1982	$L_{q^{6n}}[\frac{1}{2}, 1.414]$	337
Coppersmith 1984	$L_{q^{6n}}[\frac{1}{3}, 1.526]$	134
Joux-Lercier 2006	$L_{q^{6n}}[\frac{1}{3}, 1.442]$	126
Joux-Lercier 2006 (as revised by Shinohara et al. 2012)	$L_{q^{6n}}[\frac{1}{3}, 1.270]$	111
Joux 2012 (personal estimation)	$L_{q^{6n}}[\frac{1}{3}, 1.175]$	103
Joux 2013 (as analyzed by Adj et al. Pairing 2013)	$L_{q^{6n}}[\frac{1}{4}, 1.530]$	81
Joux-Pierrot 2014 (as analyzed by Adj et al. Waifi 2014)	$L_{q^{6n}}[\frac{1}{4}, 1.530]$	58

Recommended key sizes (circa 2013)

Security in bits	RSA $\|\|N\|\|_2$	DL: $\mathbb{F}_p$ $\|\|p\|\|_2$	DL: $\mathbb{F}_{2^m}$ m	ECC $\|\|q\|\|_2$
80	1024	1024	1500	160
112	2048	2048	3500	224
128	3072	3072	4800	256
192	7680	7680	12500	384
256	15360	15360	25000	512

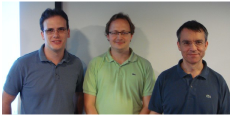
Recommended key sizes (2019)

Security in bits	RSA $\|\|N\|\|_2$	DLP: $\mathbb{F}_p$ $\|\|p\|\|_2$	~~DL: $\mathbb{F}_{2^m}$~~ $\cancel{m}$	ECC $\|\|q\|\|_2$
≈ 74	1024	1024	~~1500~~	160
≈ 106	2048	2048	~~3500~~	224
128	3072	3072	~~4800*~~	256
192	7680	7680	~~12500~~	384
256	15360	15360	~~25000~~	512

Recommended key sizes (2019)

Security in bits	RSA $\|\|N\|\|_2$	DLP: $\mathbb{F}_p$ $\|\|p\|\|_2$	~~DL: $\mathbb{F}_{2^m}$~~ ~~m~~	ECC $\|\|q\|\|_2$
≈ 74	1024	1024	~~1500~~	160
≈ 106	2048	2048	~~3500~~	224
128	3072	3072	~~4800*~~	256
192	7680	7680	~~12500~~	384
256	15360	15360	~~25000~~	512

- * Nowadays, the extension $\mathbb{F}_{2^{4800}}$ is estimated to provide a security level of around 60 bits (see [Granger-Kleinjung-Zumbrägel'18], [AMOR'16]).



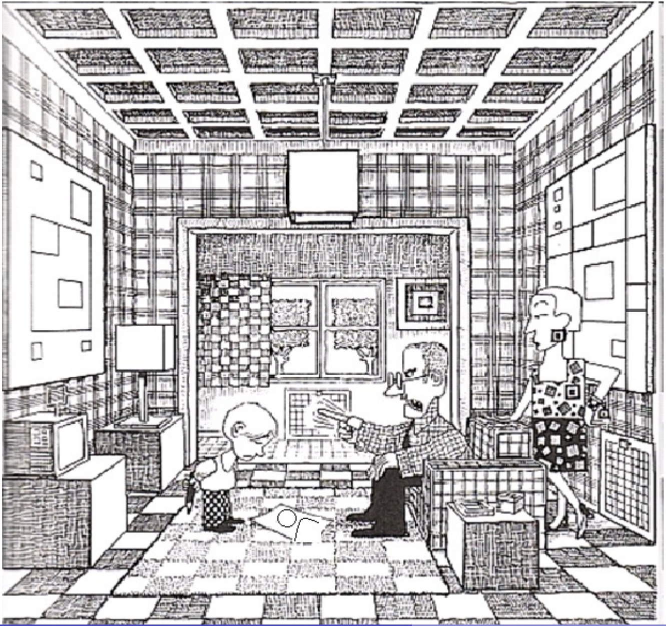
Barbulescu-Gaudry-Joux-Thomé: "A Heuristic Quasi-Polynomial Algorithm for Discrete Logarithm in Finite Fields of Small Characteristic". EUROCRYPT 2014: 1-16

Recommended key sizes (2019)

Security in bits	RSA $\|\|N\|\|_2$	DLP: $\mathbb{F}_p$ $\|\|p\|\|_2$	~~DL: $\mathbb{F}_{2^m}$ m~~	ECC $\|\|q\|\|_2$
≈ 74	1024	1024	~~1500~~	160
≈ 106	2048	2048	~~3500~~	224
128	3072	3072	~~4800*~~	256
192	7680	7680	~~12500~~	384
256	15360	15360	~~25000~~	512

- Factorization (RSA): Using the Number Field Sieve (NFS) method leads to subexponential complexity, $\approx L_N\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right]$, Where N is the RSA modulus
- DLP over $\mathbb{F}_p$: Using index-calculus methods leads to subexponential complexity, $\approx L_p\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right]$,
- ECDLP: Using the Pollard's rho method leads to exponential complexity $\sqrt{\pi \cdot q}/2$, where $q = p^k$ is the prime field extension where the elliptic curve has been defined

Elliptic-curve-based cryptography



Elliptic-curve-based cryptography

Figure: Professors Neal Koblitz and Victor Miller and a bunch of Mexican graduate students at ECC 2012 in Querétaro, México

- Elliptic-curve-based cryptography (ECC) was independently proposed by Victor Miller and Neal Koblitz in 1985.
- It took more than two decades for ECC to be widely accepted and become the most popular public-key cryptographic scheme (above its archrival RSA)
- Nowadays ECC is massively used in everyday applications

Elliptic curves

- E defined by a Weierstraß equation of the form

$$y^2 = x^3 + Ax + B$$

Elliptic curves

- E defined by a Weierstraß equation of the form

$$y^2 = x^3 + Ax + B$$

- $E(K)$ set of rational points over a field K

Elliptic curves

- E defined by a Weierstraß equation of the form

$$y^2 = x^3 + Ax + B$$

- $E(K)$ set of rational points over a field K
- Additive group law over $E(K)$

Elliptic curves

- E defined by a Weierstraß equation of the form

$$y^2 = x^3 + Ax + B$$

- $E(K)$ set of rational points over a field K
- Additive group law over $E(K)$
- Many applications in cryptography since 1985
  - EC-based Diffie-Hellman key exchange
  - EC-based Digital Signature Algorithm

Elliptic curves

- E defined by a Weierstraß equation of the form

$$y^2 = x^3 + Ax + B$$

- $E(K)$ set of rational points over a field K
- Additive group law over $E(K)$
- Many applications in cryptography since 1985
  - EC-based Diffie-Hellman key exchange
  - EC-based Digital Signature Algorithm
- Interest: smaller keys than usual cryptosystems (RSA, ElGamal, ...)

Elliptic curves

- E defined by a Weierstraß equation of the form

$$y^2 = x^3 + Ax + B$$

- $E(K)$ set of rational points over a field K
- Additive group law over $E(K)$
- Many applications in cryptography since 1985
  - EC-based Diffie-Hellman key exchange
  - EC-based Digital Signature Algorithm
- Interest: smaller keys than usual cryptosystems (RSA, ElGamal, ...)
- But there's more:
  - Bilinear pairings
  - Isogenous elliptic curves

Group cryptography

- $(\mathbb{G}_1, +)$, an additively-written cyclic group of prime order $\#\mathbb{G}_1 = \ell$

Group cryptography

- $(\mathbb{G}_1, +)$, an additively-written cyclic group of prime order $\#\mathbb{G}_1 = \ell$
- P, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$

Group cryptography

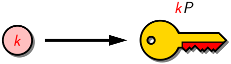
- $(\mathbb{G}_1, +)$, an additively-written cyclic group of prime order $\#\mathbb{G}_1 = \ell$
- P, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$
- Scalar multiplication: for any integer k, we have
  $kP = \underbrace{P + P + \cdots + P}_{k \text{ times}}$

Group cryptography

- $(\mathbb{G}_1, +)$, an additively-written cyclic group of prime order $\#\mathbb{G}_1 = \ell$
- P, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$
- Scalar multiplication: for any integer k, we have
  $kP = \underbrace{P + P + \cdots + P}_{k \text{ times}}$

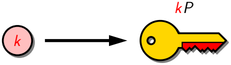
Group cryptography

- $(\mathbb{G}_1, +)$, an additively-written cyclic group of prime order $\#\mathbb{G}_1 = \ell$
- P, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$
- Scalar multiplication: for any integer k, we have $kP = \underbrace{P + P + \cdots + P}_{k \text{ times}}$



Group cryptography

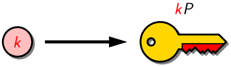
- $(\mathbb{G}_1, +)$, an additively-written cyclic group of prime order $\#\mathbb{G}_1 = \ell$
- P, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$
- Scalar multiplication: for any integer k, we have
  $kP = \underbrace{P + P + \cdots + P}_{k \text{ times}}$



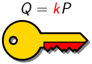
- Discrete logarithm: given $Q \in \mathbb{G}_1$, compute k such that $Q = kP$

Group cryptography

- $(\mathbb{G}_1, +)$, an additively-written cyclic group of prime order $\#\mathbb{G}_1 = \ell$
- P, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$
- Scalar multiplication: for any integer k, we have
  $kP = \underbrace{P + P + \cdots + P}_{k \text{ times}}$

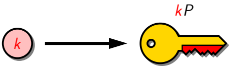


- Discrete logarithm: given $Q \in \mathbb{G}_1$, compute k such that $Q = kP$

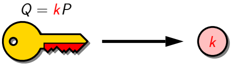


Group cryptography

- $(\mathbb{G}_1, +)$, an additively-written cyclic group of prime order $\#\mathbb{G}_1 = \ell$
- P, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$
- Scalar multiplication: for any integer k, we have
  $kP = \underbrace{P + P + \cdots + P}_{k \text{ times}}$

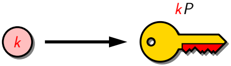


- Discrete logarithm: given $Q \in \mathbb{G}_1$, compute k such that $Q = kP$

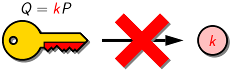


Group cryptography

- $(\mathbb{G}_1, +)$, an additively-written cyclic group of prime order $\#\mathbb{G}_1 = \ell$
- P, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$
- Scalar multiplication: for any integer k, we have
  $kP = \underbrace{P + P + \cdots + P}_{k \text{ times}}$

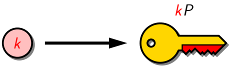


- Discrete logarithm: given $Q \in \mathbb{G}_1$, compute k such that $Q = kP$

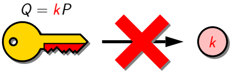


Group cryptography

- $(\mathbb{G}_1, +)$, an additively-written cyclic group of prime order $\#\mathbb{G}_1 = \ell$
- P, a generator of the group: $\mathbb{G}_1 = \langle P \rangle$
- Scalar multiplication: for any integer k, we have
  $kP = \underbrace{P + P + \cdots + P}_{k \text{ times}}$



- Discrete logarithm: given $Q \in \mathbb{G}_1$, compute k such that $Q = kP$



- We assume that the discrete logarithm problem (DLP) in $\mathbb{G}_1$ is hard

The Elliptic Curve Diffie-Hellman (ECDH) Protocol

Algorithm 1 The elliptic curve Diffie-Hellman protocol

Public parameters: Prime p, curve $E/\mathbb{F}_p$, point $P = (x, y) \in E(\mathbb{F}_p)$ of order r

Phase 1: Key pair generation

Alice

1: Select the private key $a \xleftarrow{\$} [1, r-1]$
2: Compute the public key $Q_A \leftarrow [a]P$

Bob

1: Select the private key $b \xleftarrow{\$} [1, r-1]$
2: Compute the public key $Q_B \leftarrow [b]P$

Phase 2: Shared secret computation

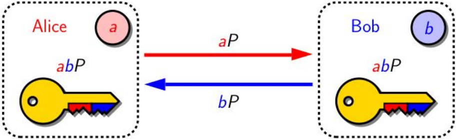
Alice

3: Send Q_A to Bob
4: Compute $R \leftarrow [a]Q_B$

Bob

3: Send Q_B to Alice
4: Compute $R \leftarrow [b]Q_A$

Final phase: The shared secret is the x-coordinate of the point R



How to efficiently compute the Elliptic Curve Diffie-Hellman (ECDH) Protocol?

The Montgomery ladder

A famous elliptic curve: Curve25519

- Curve25519 satisfies the Montgomery elliptic curve,

$$E : y^2 = x^3 + 48666 \cdot x^2 + x,$$

- Curve25519 is used for generating shared-secrets on applications such as TLS 1.3 and WhatsApp, among others.
- Proposed by Daniel J. Bernstein en 2006, it became massively popular around 2013

Daniel J. Bernstein: "Curve25519: New Diffie-Hellman Speed Records". Public Key Cryptography 2006: 207-228

Algorithm 2 Left-to-right Montgomery ladder [Montgomery'87]

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=k\cdot P}$

```
R_0 ← O; R_1 ← u_P;
for i = n − 1 downto 0 do
    if k_i = 1 then
        R_0 ← R_0 +_(P) R_1;    R_1 ← 2R_1
    else
        R_1 ← R_0 +_(P) R_1;    R_0 ← 2R_0
    end if
end for
return u_Q ← R_0
```

Algorithm 2 Left-to-right Montgomery ladder [Montgomery'87]

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=k \cdot P}$

$R_0 \leftarrow \mathcal{O}$; $R_1 \leftarrow u_P$;
for $i = n-1$ **downto** 0 **do**
  if $k_i = 1$ **then**
    $R_0 \leftarrow R_0 +_{(P)} R_1; \quad R_1 \leftarrow 2R_1$
  else
    $R_1 \leftarrow R_0 +_{(P)} R_1; \quad R_0 \leftarrow 2R_0$
  end if
end for
return $u_Q \leftarrow R_0$

Peter L. Montgomery.: "Speeding the Pollard and elliptic curve methods of factorization". Math. Comput. 48(177), 243–264 (1987)

Algorithm 2 Left-to-right Montgomery ladder [Montgomery'87]

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=k \cdot P}$

```
R_0 ← O; R_1 ← u_P;
for i = n − 1 downto 0 do
    if k_i = 1 then
        R_0 ← R_0 +_(P) R_1;    R_1 ← 2R_1
    else
        R_1 ← R_0 +_(P) R_1;    R_0 ← 2R_0
    end if
end for
return u_Q ← R_0
```

Remark 1: The Montgomery ladder maintains the invariant $R_1 - R_0 = P$ by computing at each iteration

$$(R_0, R_1) \leftarrow \begin{cases} (2R_0, 2R_0 + P), & \text{if } k_i = 0 \\ (2R_0 + P, 2R_0 + 2P), & \text{if } k_i = 1. \end{cases}$$

Algorithm 2 Left-to-right Montgomery ladder [Montgomery'87]

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=k\cdot P}$

1: $R_0 \leftarrow \mathcal{O}$; $R_1 \leftarrow u_P$;
2: **for** $i = n-1$ **downto** 0 **do**
3: **if** $k_i = 1$ **then**
4: $R_0 \leftarrow R_0 +_{(P)} R_1$; $R_1 \leftarrow 2R_1$
5: **else**
6: $R_1 \leftarrow R_0 +_{(P)} R_1$; $R_0 \leftarrow 2R_0$
7: **end if**
8: **end for**
9: **return** $u_Q \leftarrow R_0$

Remark 2: If the difference between the points R_1 and R_0 is known, it is possible to derive efficient differential addition formulas, namely,

$$U_{R_1} \leftarrow Z_P \cdot ((U_{R_1} + Z_{R_1}) \cdot (U_{R_0} - Z_{R_0}) + (U_{R_1} - Z_{R_1}) \cdot (U_{R_0} + Z_{R_0}))^2$$
$$Z_{R_1} \leftarrow u_P \cdot ((U_{R_1} + Z_{R_1}) \cdot (U_{R_0} - Z_{R_0}) - (U_{R_1} - Z_{R_1}) \cdot (U_{R_0} + Z_{R_0}))^2.$$

Using the standard trick of making $Z_P = 1$ this can be computed at a cost of $2\mathbf{m} + 1\mathbf{m_{uP}} + 2\mathbf{s} + 6\mathbf{a}$

Algorithm 2 Left-to-right Montgomery ladder [Montgomery'87]

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \dots, k_1, k_0)_2$
Ensure: $u_{Q = k \cdot P}$

$R_0 \leftarrow \mathcal{O}$; $R_1 \leftarrow u_P$;
for $i = n - 1$ **downto** 0 **do**
  if $k_i = 1$ **then**
    $R_0 \leftarrow R_0 +_{(P)} R_1$; $R_1 \leftarrow 2R_1$
  else
    $R_1 \leftarrow R_0 +_{(P)} R_1$; $R_0 \leftarrow 2R_0$
  end if
end for
return $u_Q \leftarrow R_0$

Remark 2: Similarly, the operation of doubling the point R_0, can be efficiently computed as,

$$\begin{aligned} U_{R_0} &\leftarrow (U_{R_0} + Z_{R_0})^2 \cdot (U_{R_0} - Z_{R_0})^2 \\ T &\leftarrow (U_{R_0} + Z_{R_0})^2 - (U_{R_0} - Z_{R_0})^2 \\ Z_{R_0} &\leftarrow \left[a_{24} \cdot T + (U_{R_0} - Z_{R_0})^2\right] \cdot T, \end{aligned}$$

which can be computed at a cost of $2\mathbf{m} + 1\mathbf{m}_{\mathbf{a24}} + 2\mathbf{s} + 4\mathbf{a}$, where $\mathbf{m}_{\mathbf{a24}}$ stands for one multiplication by the constant $a_{24} = \frac{A+2}{4}$.

Algorithm 2 Left-to-right Montgomery ladder [Montgomery'87]

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \dots, k_1, k_0)_2$
Ensure: $u_{Q=k\cdot P}$

1: $R_0 \leftarrow \mathcal{O}$; $R_1 \leftarrow u_P$;
2: **for** $i = n-1$ **downto** 0 **do**
3:   **if** $k_i = 1$ **then**
4:     $R_0 \leftarrow R_0 +_{(P)} R_1$; $R_1 \leftarrow 2R_1$
5:   **else**
6:     $R_1 \leftarrow R_0 +_{(P)} R_1$; $R_0 \leftarrow 2R_0$
7:   **end if**
8: **end for**
9: **return** $u_Q \leftarrow R_0$

Total computational cost: In summary, the computational cost of the Montgomery ladder is,

$$n \cdot (4\mathbf{m} + 1\mathbf{m}_{\mathbf{a24}} + 1\mathbf{m}_{\mathbf{uP}} + 4\mathbf{s} + 8\mathbf{a}) + 1\mathbf{m} + 1\mathbf{i}.$$

In the RFC 7748 [essentially] this algorithm is called X25519 (with $n = 255$)

Algorithm 3 Low-level left-to-right Montgomery ladder

Require: $P = (u_P, v_P) \in E_A/\mathbb{F}_p$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$, $a_{24} = (A+2)/4$
Ensure: $u_{Q=kP}$

Initialization: $U_{R_0} \leftarrow 1,\ Z_{R_0} \leftarrow 0,\ U_{R_1} \leftarrow u_P,\ Z_{R_1} \leftarrow 1,\ s \leftarrow 0$
for $i \leftarrow n-1$ **downto** 0 **do**
    `# timing-attack countermeasure`
    $s \leftarrow s \oplus k_i$
    $U_{R_0}, U_{R_1} \leftarrow \text{cswap}(s,\ U_{R_0},\ U_{R_1})$
    $Z_{R_0}, Z_{R_1} \leftarrow \text{cswap}(s,\ Z_{R_0},\ Z_{R_1})$
    $s \leftarrow k_i$
    `# common operations`
    $A \leftarrow U_{R_0} + Z_{R_0};\ B \leftarrow U_{R_0} - Z_{R_0}$
    `# addition`
    $C \leftarrow U_{R_1} + Z_{R_1};\ D \leftarrow U_{R_1} - Z_{R_1}$
    $C \leftarrow C \times B;\ D \leftarrow D \times A$
    $U_{R_1} \leftarrow D + C;\ U_{R_1} \leftarrow U_{R_1}^2$
    $Z_{R_1} \leftarrow D - C;\ Z_{R_1} \leftarrow Z_{R_1}^2;\ Z_{R_1} \leftarrow u_P \times Z_{R_1}$
    `# doubling`
    $A \leftarrow A^2;\ B \leftarrow B^2$
    $U_{R_0} \leftarrow A \times B$
    $A \leftarrow A - B$
    $Z_{R_0} \leftarrow a_{24} \times A;\ Z_{R_0} \leftarrow Z_{R_0} + B;\ Z_{R_0} \leftarrow Z_{R_0} \times A$
end for
$U_{R_0}, U_{R_1} \leftarrow \text{cswap}(s,\ U_{R_0},\ U_{R_1})$
$Z_{R_0}, Z_{R_1} \leftarrow \text{cswap}(s,\ Z_{R_0},\ Z_{R_1})$
$Z_{R_0} \leftarrow Z_{R_0}^{-1};\quad u_{R_0} \leftarrow U_{R_0} \times Z_{R_0}$
return $u_Q \leftarrow u_{R_0}$

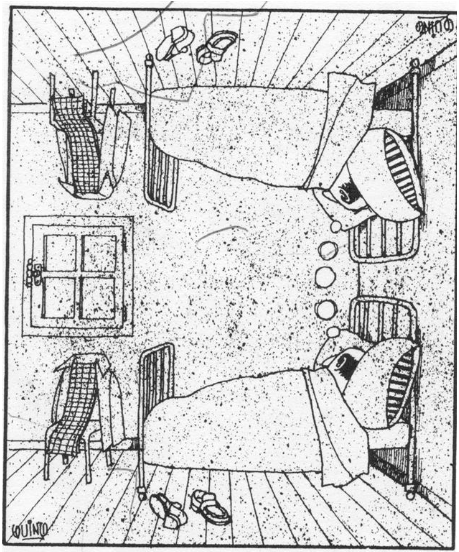
Computational cost of the X25519 and X448

- At the 128 bits of security level, the X25519 function costs

$$1021\mathbf{m} + 255\mathbf{m_{a24}} + 255\mathbf{m_{uP}} + 1020\mathbf{s} + 2040\mathbf{a} + 1\mathbf{i},$$

where each operation is performed in the prime field $\mathbb{F}_{2^{255}-19}$.

A (Pre-)computable Montgomery ladder



Algorithm 4 Right-to-left double-and-and algorithm

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $Q = k \cdot P$

1: $R_0 \leftarrow P$, $R_1 = \mathcal{O}$
2: **for** $i \leftarrow 0$ **to** $n-1$ **do**
3:     **if** $k_i = 1$ **then**
4:         $R_1 \leftarrow R_0 + R_1$
5:     **end if**
6:     $R_0 \leftarrow 2 \cdot P$
7: **end for**
8: **return** R_1

Algorithm 4 Right-to-left double-and-and algorithm [with pre-computation]

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \dots, k_1, k_0)_2$
Ensure: $Q = k \cdot P$

```
Pre-computation: Calculate and store P_i = 2^i P, for 1 ≤ i ≤ n
R_0 ← P, R_1 = O
for i ← 0 to n − 1 do
    if k_i = 1 then
        R_1 ← R_0 + R_1
    end if
    R_0 ← P_{i+1}
end for
return R_1
```

Algorithm 4 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

```
Pre-computation: Calculate and store u_{P_i}, where P_i = 2^i P, for 0 ≤ i ≤ n
Initialization: Select an order-h point S ∈ E_A(F_p)
R_0 ← u_P, R_1 ← u_S, R_2 ← u_{P−S}
for i ← 0 to n − 1 do
    if k_i = 1 then
        R_1 ← R_0 +_(R_2) R_1 (with R_2 = R_0 − R_1)
    else
        R_2 ← R_0 +_(R_1) R_2 (with R_1 = R_0 − R_2)
    end if
    R_0 ← u_{P_{i+1}}
end for
return u_Q = hR_1
```

Remark 0: This procedure only makes sense if we are in the fixed-point scenario (corresponding to the key generation phase of the DH protocol)

Algorithm 4 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

Pre-computation: Calculate and store u_{P_i}, where $P_i = 2^i P$, for $0 \leq i \leq n$
Initialization: Select an order-h point $S \in E_A(\mathbb{F}_p)$
$R_0 \leftarrow u_P$, $R_1 \leftarrow u_S$, $R_2 \leftarrow u_{P-S}$
for $i \leftarrow 0$ **to** $n-1$ **do**
  if $k_i = 1$ **then**
    $R_1 \leftarrow R_0 +_{(R_2)} R_1$ (with $R_2 = R_0 - R_1$)
  else
    $R_2 \leftarrow R_0 +_{(R_1)} R_2$ (with $R_1 = R_0 - R_2$)
  end if
  $R_0 \leftarrow u_{P_{i+1}}$
end for
return $u_Q = hR_1$

Remark 0: This procedure only makes sense if we are in the fixed-point scenario (corresponding to the key generation phase of the DH protocol) and if you are not particularly interested in recovering the y-coordinate of the output point anyway :)

Algorithm 4 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

Pre-computation: Calculate and store u_{P_i}, where $P_i = 2^i P$, for $0 \leq i \leq n$
Initialization: Select an order-h point $S \in E_A(\mathbb{F}_p)$
$R_0 \leftarrow u_P$, $R_1 \leftarrow u_S$, $R_2 \leftarrow u_{P-S}$
for $i \leftarrow 0$ **to** $n-1$ **do**
  if $k_i = 1$ **then**
    $R_1 \leftarrow R_0 +_{(R_2)} R_1$ (with $R_2 = R_0 - R_1$)
  else
    $R_2 \leftarrow R_0 +_{(R_1)} R_2$ (with $R_1 = R_0 - R_2$)
  end if
  $R_0 \leftarrow u_{P_{i+1}}$
end for
return $u_Q = hR_1$

Remark 1: R_1 must be initialized with a point $S \notin \langle P \rangle$ because the differential formulas are not complete on Montgomery curves.

Algorithm 4 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

```
Pre-computation: Calculate and store u_{P_i}, where P_i = 2^i P, for 0 ≤ i ≤ n
Initialization: Select an order-h point S ∈ E_A(F_p)
R_0 ← u_P, R_1 ← u_S, R_2 ← u_{P−S}
for i ← 0 to n − 1 do
    if k_i = 1 then
        R_1 ← R_0 +_(R_2) R_1 (with R_2 = R_0 − R_1)
    else
        R_2 ← R_0 +_(R_1) R_2 (with R_1 = R_0 − R_2)
    end if
    R_0 ← u_{P_{i+1}}
end for
return u_Q = hR_1
```

Remark 1: R_1 must be initialized with a point $S \notin \langle P \rangle$ because the differential formulas are not complete on Montgomery curves. (Really? More on this later)

Algorithm 4 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

```
Pre-computation: Calculate and store u_{P_i}, where P_i = 2^i P, for 0 ≤ i ≤ n
Initialization: Select an order-h point S ∈ E_A(F_p)
R_0 ← u_P, R_1 ← u_S, R_2 ← u_{P−S}
for i ← 0 to n − 1 do
   if k_i = 1 then
      R_1 ← R_0 +_(R_2) R_1 (with R_2 = R_0 − R_1)
   else
      R_2 ← R_0 +_(R_1) R_2 (with R_1 = R_0 − R_2)
   end if
   R_0 ← u_{P_{i+1}}
end for
return u_Q = hR_1
```

- At each iteration, the accumulator R_1 is updated in the same fashion as it would be done in a traditional right-to-left double-and-add algorithm. It follows that at the end of the main loop, $R_1 = kP + S$.
- R_2 is updated such that $R_2 = R_0 - R_1$ is always true.

Algorithm 4 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

```
Pre-computation: Calculate and store u_{P_i}, where P_i = 2^i P, for 0 ≤ i ≤ n
Initialization: Select an order-h point S ∈ E_A(F_p)
R_0 ← u_P, R_1 ← u_S, R_2 ← u_{P−S}
for i ← 0 to n − 1 do
   if k_i = 1 then
      R_1 ← R_0 +_(R_2) R_1 (with R_2 = R_0 − R_1)
   else
      R_2 ← R_0 +_(R_1) R_2 (with R_1 = R_0 − R_2)
   end if
   R_0 ← u_{P_{i+1}}
end for
return u_Q = hR_1
```

Remark 2: One can eliminate S by performing a scalar multiplication by the cofactor h, thus obtaining,
$hR_1 = h \cdot (kP + S) = hkP + hS = hkP.$

Algorithm 4 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

Pre-computation: Calculate and store u_{P_i}, where $P_i = 2^i P$, for $0 \le i \le n$
Initialization: Select an order-h point $S \in E_A(\mathbb{F}_p)$
$R_0 \leftarrow u_P$, $R_1 \leftarrow u_S$, $R_2 \leftarrow u_{P-S}$
for $i \leftarrow 0$ **to** $n-1$ **do**
  if $k_i = 1$ **then**
    $R_1 \leftarrow R_0 +_{(R_2)} R_1$ (with $R_2 = R_0 - R_1$)
  else
    $R_2 \leftarrow R_0 +_{(R_1)} R_2$ (with $R_1 = R_0 - R_2$)
  end if
  $R_0 \leftarrow u_{P_{i+1}}$
end for
return $u_Q = hR_1$

Computational cost: At the space price of allocating $n+1$ elements $u_{P_i} \in \mathbb{F}_p$, this ladder variant saves n point doubling computations as compared with the classical ladder.
Notice that this pre-computation table contains only public information. Hence, no special protection against side-channel attacks is required.

Algorithm 4 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \dots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

```
Pre-computation: Calculate and store u_{P_i}, where P_i = 2^i P, for 0 ≤ i ≤ n
Initialization: Select an order-h point S ∈ E_A(F_p)
R_0 ← u_P, R_1 ← u_S, R_2 ← u_{P−S}
for i ← 0 to n − 1 do
    if k_i = 1 then
        R_1 ← R_0 +_(R_2) R_1 (with R_2 = R_0 − R_1)
    else
        R_2 ← R_0 +_(R_1) R_2 (with R_1 = R_0 − R_2)
    end if
    R_0 ← u_{P_{i+1}}
end for
return u_Q = hR_1
```

Computational cost: However, notice that the point additions become more expensive, because in general the Z coordinate of the difference will not be equal to one anymore.
This implies that the differential point addition costs now one more field multiplication, namely, **4m** + **2s** + **6a**

Algorithm 4 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

```
Pre-computation: Calculate and store u_{P_i}, where P_i = 2^i P, for 0 ≤ i ≤ n
Initialization: Select an order-h point S ∈ E_A(F_p)
R_0 ← u_P, R_1 ← u_S, R_2 ← u_{P−S}
for i ← 0 to n − 1 do
   if k_i = 1 then
      R_1 ← R_0 +_(R_2) R_1 (with R_2 = R_0 − R_1)
   else
      R_2 ← R_0 +_(R_1) R_2 (with R_1 = R_0 − R_2)
   end if
   R_0 ← u_{P_{i+1}}
end for
return u_Q = hR_1
```

Expected time saving?: something around 30% for the X25519 function.
Question: Can we do better?

Algorithm 4 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

Pre-computation: Calculate and store u_{P_i}, where $P_i = 2^i P$, for $0 \leq i \leq n$
Initialization: Select an order-h point $S \in E_A(\mathbb{F}_p)$
$R_0 \leftarrow u_P$, $R_1 \leftarrow u_S$, $R_2 \leftarrow u_{P-S}$
for $i \leftarrow 0$ **to** $n-1$ **do**
  if $k_i = 1$ **then**
    $R_1 \leftarrow R_0 +_{(R_2)} R_1$ (with $R_2 = R_0 - R_1$)
  else
    $R_2 \leftarrow R_0 +_{(R_1)} R_2$ (with $R_1 = R_0 - R_2$)
  end if
  $R_0 \leftarrow u_{P_{i+1}}$
end for
return $u_Q = hR_1$

A closer look shows that we can express the differential point addition of $R_3 = R_0 +_{(R_2)} R_1$ as,

$$U_{R_3} \leftarrow Z_{R_2}((U_{R_1} + Z_{R_1}) + \mu(U_{R_1} - Z_{R_1}))^2$$
$$Z_{R_3} \leftarrow U_{R_2}((U_{R_1} + Z_{R_1}) - \mu(U_{R_1} - Z_{R_1}))^2,$$

where, $\mu = \dfrac{u_{R_0} + 1}{u_{R_0} - 1}$. The above differential point addition formula can be computed at a cost of **3m** $+$ **2s** $+$ **6a**

Algorithm 5 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

Pre-computation: Calculate and store u_{P_i}, where $P_i = 2^i P$, for $0 \leq i \leq n$
Initialization: Select an order-h point $S \in E_A(\mathbb{F}_p)$
$R_0 \leftarrow u_P$, $R_1 \leftarrow u_S$, $R_2 \leftarrow u_{P-S}$
for $i \leftarrow 0$ **to** $n-1$ **do**
  if $k_i = 1$ **then**
    $R_1 \leftarrow R_0 +_{(R_2)} R_1$ (with $R_2 = R_0 - R_1$)
  else
    $R_2 \leftarrow R_0 +_{(R_1)} R_2$ (with $R_1 = R_0 - R_2$)
  end if
  $R_0 \leftarrow u_{P_{i+1}}$
end for
return $u_Q = hR_1$

Algorithm 5 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p),\ k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

Pre-computation: Calculate and store $\mu_i = \frac{u_{P_i}+1}{u_{P_i}-1}$, where $P_i = 2^i P$, for $0 \leq i \leq n$
Initialization: Select an order-h point $S \in E_A(\mathbb{F}_p)$
$R_0 \leftarrow u_P,\ R_1 \leftarrow u_S,\ R_2 \leftarrow u_{P-S}$
for $i \leftarrow 0$ **to** $n-1$ **do**
    if $k_i = 1$ **then**
        $R_1 \leftarrow R_0 +_{(R_2)} R_1$ (with $R_2 = R_0 - R_1$)
    else
        $R_2 \leftarrow R_0 +_{(R_1)} R_2$ (with $R_1 = R_0 - R_2$)
    end if
    $R_0 \leftarrow u_{P_{i+1}}$
end for
return $u_Q = hR_1$

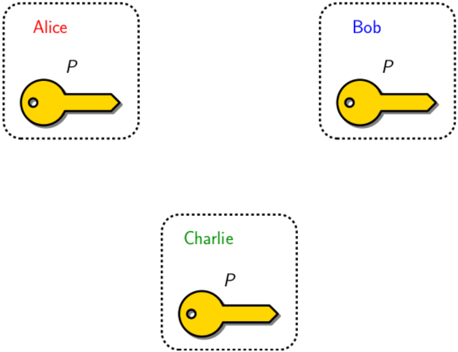
Algorithm 5 Right-to-left Montgomery ladder

Require: $P = (u_P, v_P) \in E_A(\mathbb{F}_p)$, $k = (k_{n-1} = 1, k_{n-2}, \ldots, k_1, k_0)_2$
Ensure: $u_{Q=hkP}$

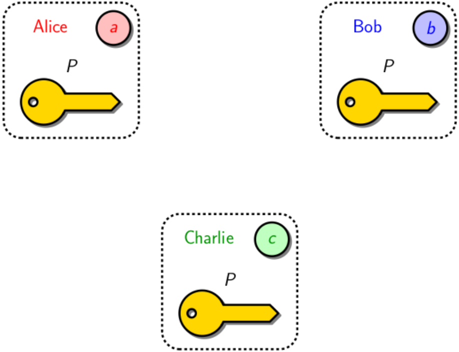
Pre-computation: Calculate and store $\mu_i = \frac{u_{P_i}+1}{u_{P_i}-1}$, where $P_i = 2^i P$, for $0 \leq i \leq n$
Initialization: Select an order-h point $S \in E_A(\mathbb{F}_p)$
$R_0 \leftarrow u_P$, $R_1 \leftarrow u_S$, $R_2 \leftarrow u_{P-S}$
for $i \leftarrow 0$ **to** $n-1$ **do**
  if $k_i = 1$ **then**
    $R_1 \leftarrow R_0 +_{(R_2)} R_1$ (with $R_2 = R_0 - R_1$)
  else
    $R_2 \leftarrow R_0 +_{(R_1)} R_2$ (with $R_1 = R_0 - R_2$)
  end if
  $R_0 \leftarrow u_{P_{i+1}}$
end for
return $u_Q = hR_1$

Assuming that the architecture is byte-addressable, the memory space required for the X25519 function is,
$(255 - 3) \cdot 32B \approx 8KB$,
while in the X448 function setting, we need,
$(448 - 2) \cdot 56B \approx 25KB$.

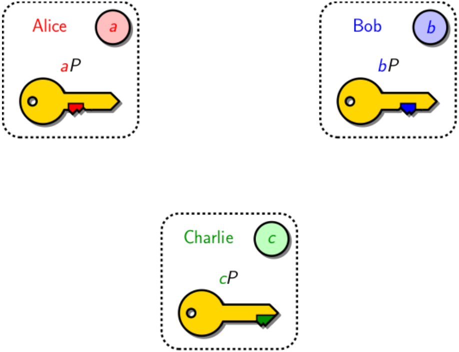
Design problem: How to establish a one-round tripartite shared-secret protocol?



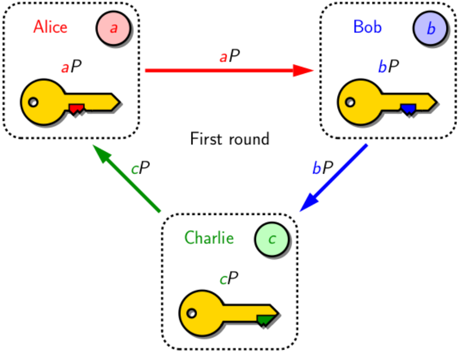
Design problem: How to establish a one-round tripartite shared-secret protocol?



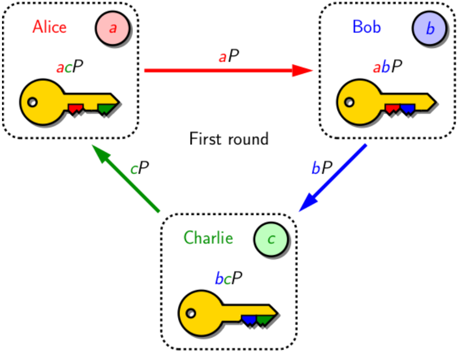
Design problem: How to establish a one-round tripartite shared-secret protocol?



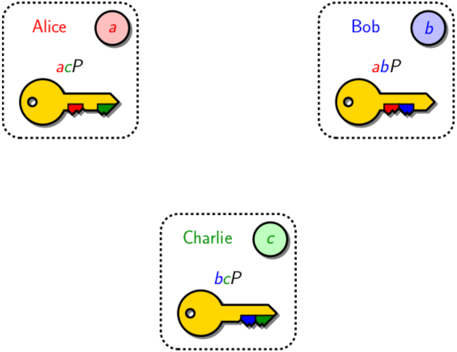
Design problem: How to establish a one-round tripartite shared-secret protocol?



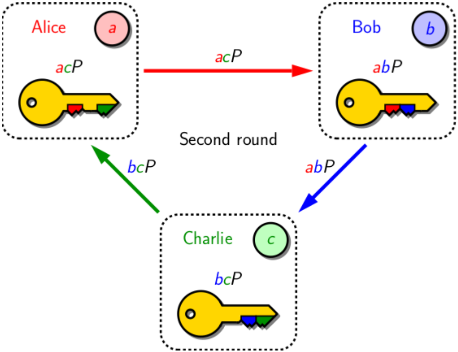
Design problem: How to establish a one-round tripartite shared-secret protocol?



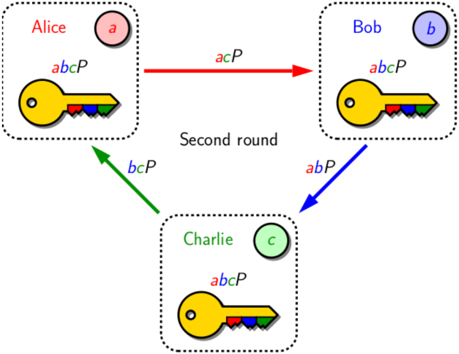
Design problem: How to establish a one-round tripartite shared-secret protocol?



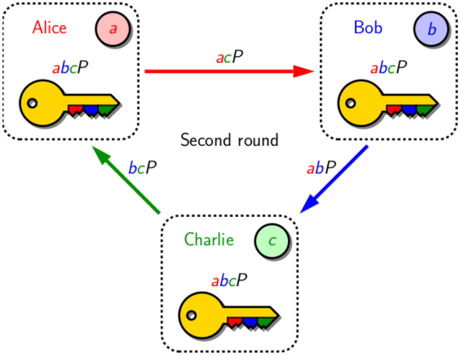
Design problem: How to establish a one-round tripartite shared-secret protocol?



Design problem: How to establish a one-round tripartite shared-secret protocol?



Design problem: How to establish a one-round tripartite shared-secret protocol?



- This problem remained open since the 1976 Diffie-Hellman paper,: There exists a tripartite Diffie-Hellman protocol that can be executed in just one round of public key exchanges?

Bilinear pairings

- $(\mathbb{G}_2, \times)$, a multiplicatively-written cyclic group of order $\#\mathbb{G}_2 = \#\mathbb{G}_1 = \ell$

Bilinear pairings

- $(\mathbb{G}_2, \times)$, a multiplicatively-written cyclic group of order $\#\mathbb{G}_2 = \#\mathbb{G}_1 = \ell$
- A bilinear pairing on $(\mathbb{G}_1, \mathbb{G}_2)$ is a map

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$$

that satisfies the following conditions:

Bilinear pairings

- $(\mathbb{G}_2, \times)$, a multiplicatively-written cyclic group of order $\#\mathbb{G}_2 = \#\mathbb{G}_1 = \ell$
- A bilinear pairing on $(\mathbb{G}_1, \mathbb{G}_2)$ is a map

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$$

that satisfies the following conditions:

  - non-degeneracy: $\hat{e}(P, P) \neq 1_{\mathbb{G}_2}$ (equivalently $\hat{e}(P, P)$ generates $\mathbb{G}_2$)

Bilinear pairings

- $(\mathbb{G}_2, \times)$, a multiplicatively-written cyclic group of order $\#\mathbb{G}_2 = \#\mathbb{G}_1 = \ell$
- A bilinear pairing on $(\mathbb{G}_1, \mathbb{G}_2)$ is a map

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$$

that satisfies the following conditions:

  - non-degeneracy: $\hat{e}(P, P) \neq 1_{\mathbb{G}_2}$ (equivalently $\hat{e}(P, P)$ generates $\mathbb{G}_2$)
  - bilinearity:
    $\hat{e}(Q_1 + Q_2, R) = \hat{e}(Q_1, R) \cdot \hat{e}(Q_2, R) \quad \hat{e}(Q, R_1 + R_2) = \hat{e}(Q, R_1) \cdot \hat{e}(Q, R_2)$

Bilinear pairings

- $(\mathbb{G}_2, \times)$, a multiplicatively-written cyclic group of order $\#\mathbb{G}_2 = \#\mathbb{G}_1 = \ell$
- A bilinear pairing on $(\mathbb{G}_1, \mathbb{G}_2)$ is a map

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$$

that satisfies the following conditions:

- non-degeneracy: $\hat{e}(P, P) \neq 1_{\mathbb{G}_2}$ (equivalently $\hat{e}(P, P)$ generates $\mathbb{G}_2$)
- bilinearity:
  $\hat{e}(Q_1 + Q_2, R) = \hat{e}(Q_1, R) \cdot \hat{e}(Q_2, R) \quad \hat{e}(Q, R_1 + R_2) = \hat{e}(Q, R_1) \cdot \hat{e}(Q, R_2)$
- computability: $\hat{e}$ can be efficiently computed

Bilinear pairings

- $(\mathbb{G}_2, \times)$, a multiplicatively-written cyclic group of order $\#\mathbb{G}_2 = \#\mathbb{G}_1 = \ell$
- A bilinear pairing on $(\mathbb{G}_1, \mathbb{G}_2)$ is a map

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$$

  that satisfies the following conditions:
  - non-degeneracy: $\hat{e}(P, P) \neq 1_{\mathbb{G}_2}$ (equivalently $\hat{e}(P, P)$ generates $\mathbb{G}_2$)
  - bilinearity:
    $\hat{e}(Q_1+Q_2, R) = \hat{e}(Q_1, R)\cdot\hat{e}(Q_2, R) \quad \hat{e}(Q, R_1+R_2) = \hat{e}(Q, R_1)\cdot\hat{e}(Q, R_2)$
  - computability: $\hat{e}$ can be efficiently computed
- Immediate property: for any two integers k_1 and k_2

$$\hat{e}(k_1 Q, k_2 R) = \hat{e}(Q, R)^{k_1 k_2}$$

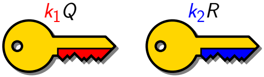
Bilinear pairings

- $(\mathbb{G}_2, \times)$, a multiplicatively-written cyclic group of order $\#\mathbb{G}_2 = \#\mathbb{G}_1 = \ell$
- A bilinear pairing on $(\mathbb{G}_1, \mathbb{G}_2)$ is a map

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$$

  that satisfies the following conditions:
  - non-degeneracy: $\hat{e}(P, P) \neq 1_{\mathbb{G}_2}$ (equivalently $\hat{e}(P, P)$ generates $\mathbb{G}_2$)
  - bilinearity:
    $\hat{e}(Q_1+Q_2, R) = \hat{e}(Q_1, R)\cdot\hat{e}(Q_2, R) \quad \hat{e}(Q, R_1+R_2) = \hat{e}(Q, R_1)\cdot\hat{e}(Q, R_2)$
  - computability: $\hat{e}$ can be efficiently computed
- Immediate property: for any two integers k_1 and k_2

$$\hat{e}(k_1 Q, k_2 R) = \hat{e}(Q, R)^{k_1 k_2}$$



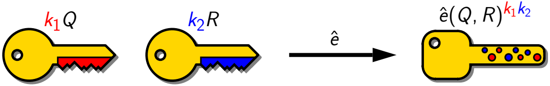
Bilinear pairings

- $(\mathbb{G}_2, \times)$, a multiplicatively-written cyclic group of order $\#\mathbb{G}_2 = \#\mathbb{G}_1 = \ell$
- A bilinear pairing on $(\mathbb{G}_1, \mathbb{G}_2)$ is a map

$$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$$

  that satisfies the following conditions:
  - non-degeneracy: $\hat{e}(P, P) \neq 1_{\mathbb{G}_2}$ (equivalently $\hat{e}(P, P)$ generates $\mathbb{G}_2$)
  - bilinearity:
    $\hat{e}(Q_1 + Q_2, R) = \hat{e}(Q_1, R) \cdot \hat{e}(Q_2, R) \quad \hat{e}(Q, R_1 + R_2) = \hat{e}(Q, R_1) \cdot \hat{e}(Q, R_2)$
  - computability: $\hat{e}$ can be efficiently computed
- Immediate property: for any two integers k_1 and k_2

$$\hat{e}(k_1 Q, k_2 R) = \hat{e}(Q, R)^{k_1 k_2}$$



Pairings in cryptography

- At first, used to attack supersingular elliptic curves
  - Menezes-Okamoto-Vanstone and Frey-Rück attacks, 1993 and 1994

$$\begin{array}{ccc} \mathrm{DLP}_{\mathbb{G}_1} & <_{\mathrm{P}} & \mathrm{DLP}_{\mathbb{G}_2} \\ kP & \longrightarrow & \hat{e}(kP, P) = \hat{e}(P, P)^k \end{array}$$

  - for cryptographic applications, we will also require the DLP in $\mathbb{G}_2$ to be hard

Pairings in cryptography

- At first, used to attack supersingular elliptic curves
  - Menezes-Okamoto-Vanstone and Frey-Rück attacks, 1993 and 1994

$$\mathrm{DLP}_{\mathbb{G}_1} \quad <_{\mathrm{P}} \quad \mathrm{DLP}_{\mathbb{G}_2}$$

$$kP \quad \longrightarrow \quad \hat{e}(kP, P) = \hat{e}(P, P)^k$$

  - for cryptographic applications, we will also require the DLP in $\mathbb{G}_2$ to be hard
- One-round three-party key agreement (Joux, 2000)

Pairings in cryptography

- At first, used to attack supersingular elliptic curves
  - Menezes-Okamoto-Vanstone and Frey-Rück attacks, 1993 and 1994

$$\begin{array}{ccc} \mathrm{DLP}_{\mathbb{G}_1} & <_{\mathrm{P}} & \mathrm{DLP}_{\mathbb{G}_2} \\ kP & \longrightarrow & \hat{e}(kP, P) = \hat{e}(P, P)^k \end{array}$$

  - for cryptographic applications, we will also require the DLP in $\mathbb{G}_2$ to be hard
- One-round three-party key agreement (Joux, 2000)
- Identity-based encryption
  - Boneh–Franklin, 2001
  - Sakai–Kasahara, 2001

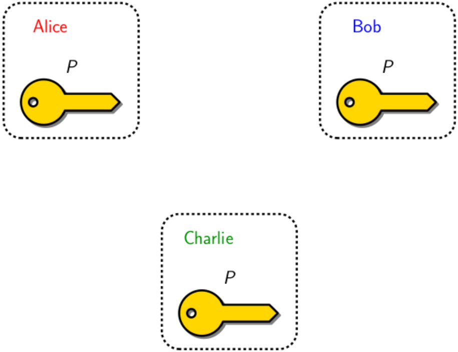
Pairings in cryptography

- At first, used to attack supersingular elliptic curves
  - Menezes-Okamoto-Vanstone and Frey-Rück attacks, 1993 and 1994

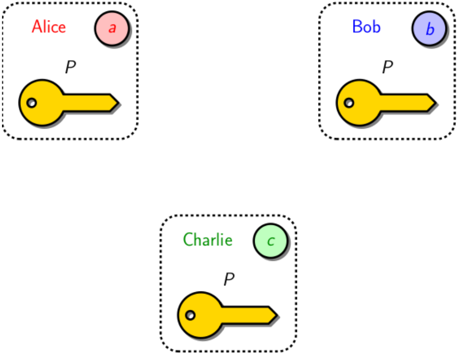
$$\begin{array}{ccc} \mathrm{DLP}_{\mathbb{G}_1} & <_{\mathrm{P}} & \mathrm{DLP}_{\mathbb{G}_2} \\ kP & \longrightarrow & \hat{e}(kP, P) = \hat{e}(P, P)^k \end{array}$$

  - for cryptographic applications, we will also require the DLP in $\mathbb{G}_2$ to be hard
- One-round three-party key agreement (Joux, 2000)
- Identity-based encryption
  - Boneh–Franklin, 2001
  - Sakai–Kasahara, 2001
- Short digital signatures, Aggregate signatures
  - Boneh–Lynn–Shacham, 2001
  - Boneh–Gentry–Lynn–Shacham, 2004
- cryptocurrencies, Pinocchio, Zcash 2013

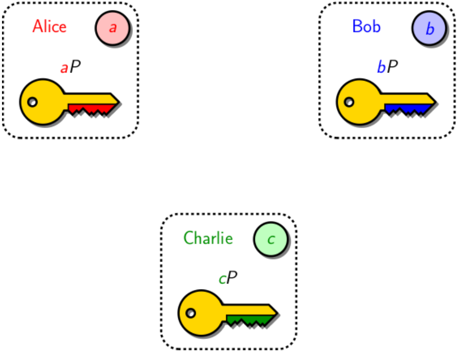
Design problem: How to establish a one-round tripartite shared-secret protocol? Solution: A One Round Protocol for Tripartite Diffie-Hellman.



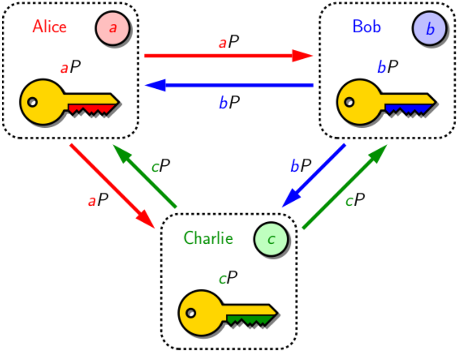
Design problem: How to establish a one-round tripartite shared-secret protocol? Solution:A One Round Protocol for Tripartite Diffie-Hellman.



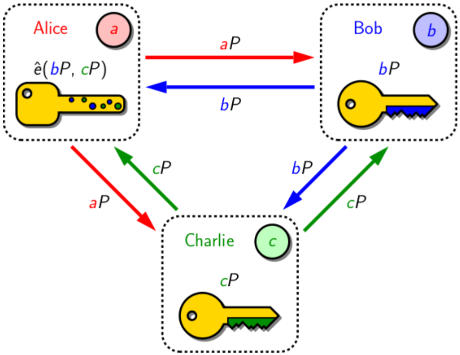
Design problem: How to establish a one-round tripartite shared-secret protocol? Solution:A One Round Protocol for Tripartite Diffie-Hellman.



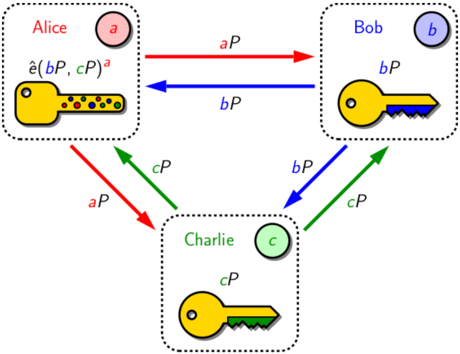
Design problem: How to establish a one-round tripartite shared-secret protocol? Solution: A One Round Protocol for Tripartite Diffie-Hellman.



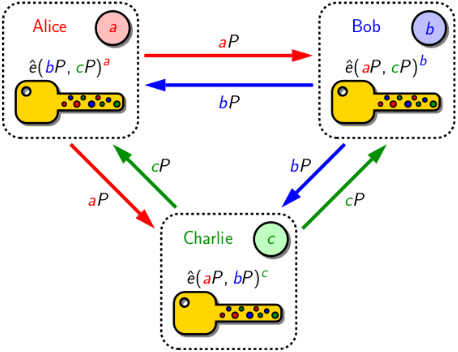
Design problem: How to establish a one-round tripartite shared-secret protocol? Solution: A One Round Protocol for Tripartite Diffie-Hellman.



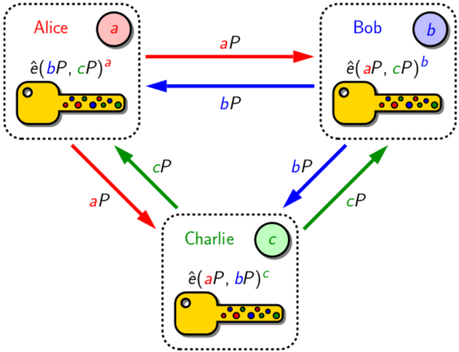
Design problem: How to establish a one-round tripartite shared-secret protocol? Solution: A One Round Protocol for Tripartite Diffie-Hellman.



Design problem: How to establish a one-round tripartite shared-secret protocol? Solution:A One Round Protocol for Tripartite Diffie-Hellman.

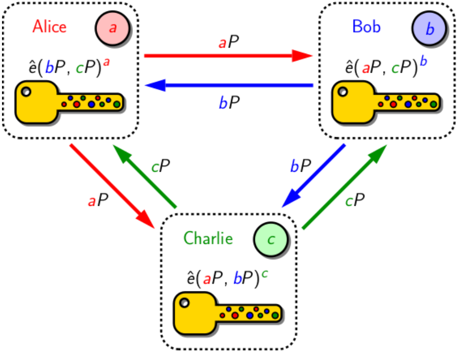


Design problem: How to establish a one-round tripartite shared-secret protocol? Solution: A One Round Protocol for Tripartite Diffie-Hellman.



- This problem remained open since the 1976 Diffie-Hellman paper,: There exists a tripartite Diffie-Hellman protocol that can be executed in just one round of public key exchanges?

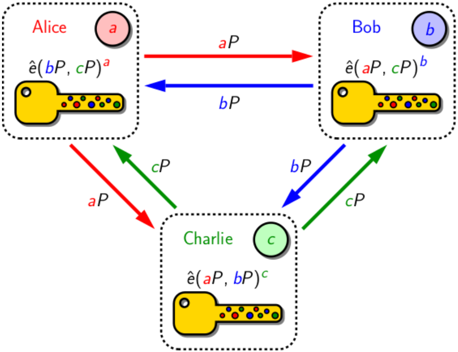
Design problem: How to establish a one-round tripartite shared-secret protocol? Solution: A One Round Protocol for Tripartite Diffie-Hellman.



- The protocol works because of,

$$\hat{e}(bP, cP)^a = \hat{e}(aP, cP)^b = \hat{e}(aP, bP)^c = \hat{e}(P, P)^{abc}$$

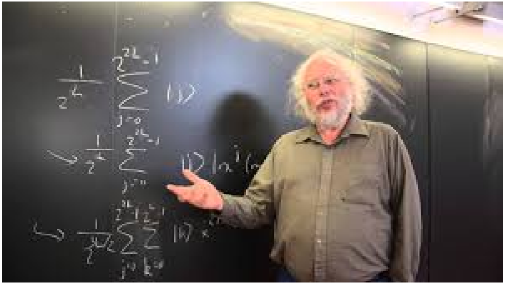
Design problem: How to establish a one-round tripartite shared-secret protocol? Solution:A One Round Protocol for Tripartite Diffie-Hellman.



R. Sakai, K. Oghishi, and M. Kasahara. "Cryptosystems based on pairing". SCIS2000: 26–28, January 2000
Antoine Joux: "A One Round Protocol for Tripartite Diffie-Hellman". ANTS 2000: 385-394
S. Mitsunari, R. Sakai, and M. Kasahara. A new traitor tracing. IEICE Trans. Fundamentals , E85A(2):481–484, Feb 2002

[Apocalyptic] scenario for the next years: The arrival of large-scale quantum computers

[Apocalyptic] scenario for the next years: The arrival of large-scale quantum computers

-  

  - A quantum computer implementation of Peter Shor algorithm for factorization of integer numbers will imply that the computational effort for breaking elliptic-curve discrete logs will become polynomial.
  - In practice, this means that breaking commercial [EC]DLP would go from billions of years to hundred of hours.

[Apocalyptic] scenario for the next years: The arrival of large-scale quantum computers

-  

Along with ECC, RSA and DSA public key crypto-schemes will also go to extinction

Design problem: How to construct a post-quantum Diffie-Hellman protocol?

Answers against the [Apocalyptic] scenario: Post-Quantum Cryptography (PQC)

- About two years ago, NIST launched a Post-Quantum Cryptography (PQC) standardization contest. NIST stated that

  'regardless of whether we can estimate the exact time of the arrival of the quantum computing era, we must begin now to prepare our information security systems to be able to resist quantum computing."

- The main focus of the contest is to find new PQC signature/verification and shared key establishment protocols. The latter task should be done using a scheme known as Key Encapsulation Mechanism (KEM).

Answers against the [Apocalyptic] scenario: Post-Quantum Cryptography (PQC)

- Out of 82 initial candidates only seven advanced to the third round, whereas another eight were declared alternative candidates. The surviving candidates can be classified into five main categories.
  - Lattice-based cryptography
  - Code-based crypto
  - Multivariate-based crypto
  - hash-based crypto
  - isogeny-based crypto

Design problem: How to construct a post-quantum Diffie-Hellman protocol using isogeny-based crypto?

[More] Mathematical definitions: recap

An *Elliptic Curve* in Weierstrass short model over a finite field $\mathbb{F}_q$ where $q = p^m$ for some prime $p > 3$, is given by the equation

$$E/\mathbb{F}_q : Y^2 = X^3 + AX + B$$

where $A, B \in \mathbb{F}_q$.

The *j-invariant* $j(E)$ of a curve acts like a fingerprint of a curve and it is given by

$$j(E) = \frac{1728 \cdot 4A^2}{4A^2 + 27B^2}.$$

A point P in $E(\mathbb{F}_q)$ is a pair (x, y) such that $x^3 + Ax + B - y^2 = 0$.

[More] Mathematical definitions: recap

- We can **Add** points

$$R := P + Q,$$

- **Double** a point

$$[2]P := P + P$$

- and multiply by a scalar as,

$$[m]P := P + P + \cdots + P, (m-1)(\text{times}).$$

- The minimum integer m such that $[m]P = \mathcal{O}$ is called the **order** of P.
- The **subgroup generated** by P is the set $\{P, [2]P, [3]P, \ldots, [m-1]P, \mathcal{O}\}$ and is denoted by $\langle P \rangle$.
- The m-**torsion subgroup** is defined as $E[m] = \{P \in E \mid [m]P = \mathcal{O}\}$.

[More] Mathematical definitions: recap

- (Hasse's Theorem)The number of rational points in an elliptic curve is bounded by

$$\#E(\mathbb{F}_q) = q + 1 - t, \qquad | \, t \, | \leq 2\sqrt{q}.$$

- E is supersingular if $p|t$, i.e., if

$$\#E(\mathbb{F}_q) = q + 1 \bmod p.$$

Otherwise E is said to be ordinary.

Basic definitions of isogenies

- An *Isogeny* $\phi : E \to E'$ is an homomorphism between elliptic curves given by rational functions. Given P and Q in E_0 it follows that
  - $\phi(P+Q) = \phi(P) + \phi(Q)$,
  - $\phi(\mathcal{O}) = \mathcal{O}$.
- The *Kernel* of an Isogeny ϕ is the set

$$K = \{P \in E \mid \phi(P) = \mathcal{O}\}.$$

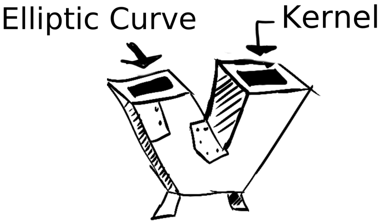
  Note: In this talk the degree of an isogeny is $s := \#K$.

- Let E and E' be two elliptic curves defined over $\mathbb{F}_q$. If there exists an isogeny $\phi : E \to E'$, then we say that E and E' are isogenous.
- If two elliptic curves E and E' are isogenous over $\mathbb{F}_q$, either both of them are supersingular or both of them are ordinary.

Basic definitions of isogenies

- Let E be an elliptic curve and $P \in E$ be an order m point.
- Then there exists an elliptic curve E' and an isogeny $\phi_P : E \to E'$ such that the *Kernel* of ϕ_P is $K = \langle P \rangle$, *i.e.* $\phi_P(Q) = \mathcal{O}$ for each $Q \in \langle P \rangle$. We write

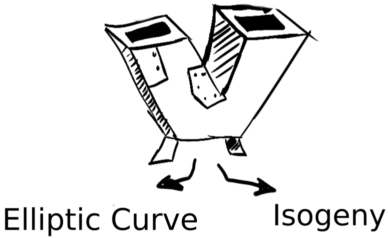
$$E' = E/\langle P \rangle$$



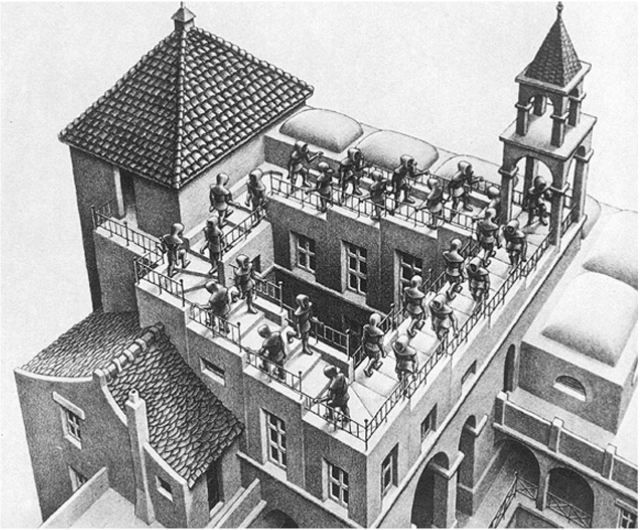
Basic definitions of isogenies

- Let E be an elliptic curve and $P \in E$ be an order m point.
- Then there exists an elliptic curve E' and an isogeny $\phi_P : E \to E'$ such that the *Kernel* of ϕ_P is $K = \langle P \rangle$, *i.e.* $\phi_P(Q) = \mathcal{O}$ for each $Q \in \langle P \rangle$. We write

$$E' = E/\langle P \rangle$$



Design problem: How to construct a post-quantum Diffie-Hellman protocol using isogeny-based crypto?



Diffie-Hellman like protocol using isogenies: The SIDH protocol [de Feo-Jao 2011]

SIDH framework:

- Find a prime p of the form $p = 2^{e_A} \cdot 3^{e_B} - 1$,
- Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$.
- $E[2^{e_A}](\mathbb{F}_{p^2}) = \langle P_A, Q_A \rangle$ and $E[3^{e_B}](\mathbb{F}_{p^2}) = \langle P_B, Q_B \rangle$.

Diffie-Hellman like protocol using isogenies: The SIDH protocol [de Feo-Jao 2011]

SIDH framework:

- Find a prime p of the form $p = 2^{e_A} \cdot 3^{e_B} - 1$,
- Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$.
- $E[2^{e_A}](\mathbb{F}_{p^2}) = \langle P_A, Q_A \rangle$ and $E[3^{e_B}](\mathbb{F}_{p^2}) = \langle P_B, Q_B \rangle$.

General description of the SIDH protocol

E

$E/\langle R_A, R_B \rangle$

Diffie-Hellman like protocol using isogenies: The SIDH protocol [de Feo-Jao 2011]

SIDH framework:

- Find a prime p of the form $p = 2^{e_A} \cdot 3^{e_B} - 1$,
- Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$.
- $E[2^{e_A}](\mathbb{F}_{p^2}) = \langle P_A, Q_A \rangle$ and $E[3^{e_B}](\mathbb{F}_{p^2}) = \langle P_B, Q_B \rangle$.

General description of the SIDH protocol

$$R_A \leftarrow [n_A]P_A + [m_A]Q_A$$
$$R_B \leftarrow [n_B]P_B + [m_B]Q_B$$

$$\begin{tikzcd}
E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle R_A \rangle \\
E/\langle R_B \rangle & E/\langle R_A, R_B \rangle
\end{tikzcd}$$

Diffie-Hellman like protocol using isogenies: The SIDH protocol [de Feo-Jao 2011]

SIDH framework:

- Find a prime p of the form $p = 2^{e_A} \cdot 3^{e_B} - 1$,
- Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$.
- $E[2^{e_A}](\mathbb{F}_{p^2}) = \langle P_A, Q_A \rangle$ and $E[3^{e_B}](\mathbb{F}_{p^2}) = \langle P_B, Q_B \rangle$.

General description of the SIDH protocol

$$R_A \leftarrow [n_A]P_A + [m_A]Q_A$$
$$R_B \leftarrow [n_B]P_B + [m_B]Q_B$$

$$\begin{tikzcd}
E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle R_A \rangle \arrow[dl, dashed, "{\phi_A(P_B),\phi_A(Q_B),E/\langle R_A \rangle}"'] \\
E/\langle R_B \rangle & E/\langle R_A, R_B \rangle
\end{tikzcd}$$

Diffie-Hellman like protocol using isogenies: The SIDH protocol [de Feo-Jao 2011]

SIDH framework:

- Find a prime p of the form $p = 2^{e_A} \cdot 3^{e_B} - 1$,
- Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$.
- $E[2^{e_A}](\mathbb{F}_{p^2}) = \langle P_A, Q_A \rangle$ and $E[3^{e_B}](\mathbb{F}_{p^2}) = \langle P_B, Q_B \rangle$.

General description of the SIDH protocol

$$R_A \leftarrow [n_A]P_A + [m_A]Q_A$$
$$R_B \leftarrow [n_B]P_B + [m_B]Q_B$$

$$\begin{tikzcd}
E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle R_A \rangle \arrow[dl, dashed, bend right=10, "{\phi_A(P_B),\phi_A(Q_B),E/\langle R_A\rangle}"'] \\
E/\langle R_B \rangle \arrow[ur, dashed, bend right=10, "{\phi_B(P_A),\phi_B(Q_A),E/\langle R_B\rangle}"'] & E/\langle R_A, R_B \rangle
\end{tikzcd}$$

Diffie-Hellman like protocol using isogenies: The SIDH protocol [de Feo-Jao 2011]

SIDH framework:

- Find a prime p of the form $p = 2^{e_A} \cdot 3^{e_B} - 1$,
- Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$.
- $E[2^{e_A}](\mathbb{F}_{p^2}) = \langle P_A, Q_A \rangle$ and $E[3^{e_B}](\mathbb{F}_{p^2}) = \langle P_B, Q_B \rangle$.

General description of the SIDH protocol

$$\phi_B(R_A) \leftarrow [n_A]\phi_B(P_A) + [m_A]\phi_B(Q_A)$$
$$\phi_A(R_B) \leftarrow [n_B]\phi_A(P_B) + [m_B]\phi_A(Q_B)$$

$$\begin{tikzcd}
E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle R_A \rangle \arrow[d, "\phi'_B"] \arrow[dl, dashed, bend right=10, "{\phi_A(P_B),\phi_A(Q_B),E/\langle R_A\rangle}"'] \\
E/\langle R_B \rangle \arrow[r, "\phi'_A"'] \arrow[ur, dashed, bend right=10, "{\phi_B(P_A),\phi_B(Q_A),E/\langle R_B\rangle}"'] & E/\langle R_A, R_B \rangle
\end{tikzcd}$$

Diffie-Hellman like protocol using isogenies: The SIDH protocol [de Feo-Jao 2011]

SIDH framework:

- Find a prime p of the form $p = 2^{e_A} \cdot 3^{e_B} - 1$,
- Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with $\#E(\mathbb{F}_{p^2}) = (p+1)^2$.
- $E[2^{e_A}](\mathbb{F}_{p^2}) = \langle P_A, Q_A \rangle$ and $E[3^{e_B}](\mathbb{F}_{p^2}) = \langle P_B, Q_B \rangle$.

General description of the SIDH protocol

$$\phi_B(R_A) \leftarrow [n_A]\phi_B(P_A) + [m_A]\phi_B(Q_A)$$
$$\phi_A(R_B) \leftarrow [n_B]\phi_A(P_B) + [m_B]\phi_A(Q_B)$$

$$\begin{tikzcd}
E \arrow[r, "\phi_A"] \arrow[d, "\phi_B"'] & E/\langle R_A \rangle \arrow[d, "\phi'_B"] \arrow[dl, dashed, bend right=10, "{\phi_A(P_B), \phi_A(Q_B), E/\langle R_A \rangle}"'] \\
E/\langle R_B \rangle \arrow[r, "\phi'_A"'] \arrow[ur, dashed, bend right=10, "{\phi_B(P_A), \phi_B(Q_A), E/\langle R_B \rangle}"'] & E/\langle R_A, R_B \rangle
\end{tikzcd}$$

where the shared secret key is the j-invariant $j(E/\langle R_A, R_B \rangle)$.

Design problem: How to construct a post-quantum Diffie-Hellman protocol?

Overviewing the CSIDH [Castryck-Lange-Martindale-Panny-Renes Asiacrypt'18]

Public parameter:
$E/\mathbb{F}_p \colon By^2 = x^3 + Ax^2 + x,$

Alice | Bob

$e_A = (e_1, \ldots, e_n) \xleftarrow{\$} [\![-m \mathinner{..} m]\!]^n$ | $e_B = (f_1, \ldots, f_n) \xleftarrow{\$} [\![-m \mathinner{..} m]\!]^n$

$E_A = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E$ $\xrightarrow{E_A}$ $E_B = \mathfrak{l}_1^{f_1} * \cdots * \mathfrak{l}_n^{f_n} * E$

$\xleftarrow{E_B}$

$E_{ba} = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_B$ | $E_{ab} = \mathfrak{l}_1^{f_1} * \cdots * \mathfrak{l}_n^{f_n} * E_A$

Figure: CSIDH key-exchange protocol

CSIDH works over a finite field $\mathbb{F}_p$, where p is a prime of the form

$$p := 4 \prod_{i=1}^{n} \ell_i - 1$$

Overviewing the CSIDH [Castryck-Lange-Martindale-Panny-Renes Asiacrypt'18]

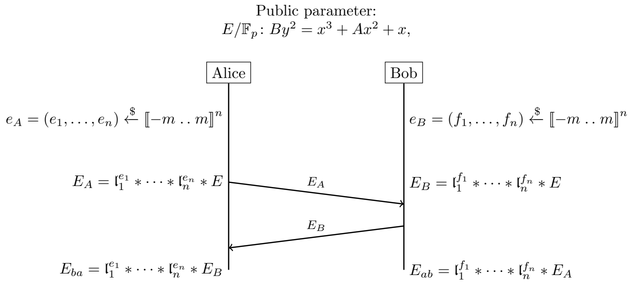
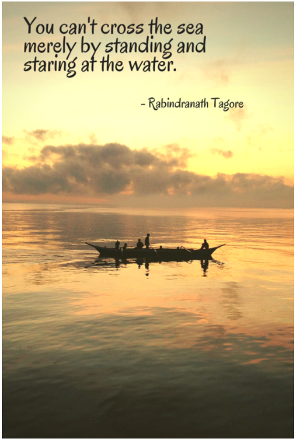


Figure: CSIDH key-exchange protocol

$(p+1)/4 = 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 \cdot 59 \cdot 61 \cdot 67 \cdot 71 \cdot 73 \cdot 79 \cdot 83 \cdot \mathbf{89} \cdot 97 \cdot 101 \cdot 103 \cdot 107 \cdot 109 \cdot 113 \cdot 127 \cdot 131 \cdot 137 \cdot 139 \cdot 149 \cdot 151 \cdot 157 \cdot 163 \cdot 167 \cdot 173 \cdot 179 \cdot 181 \cdot 191 \cdot 193 \cdot 197 \cdot 199 \cdot 211 \cdot 223 \cdot 227 \cdot 229 \cdot 233 \cdot 239 \cdot 241 \cdot 251 \cdot 257 \cdot 263 \cdot 269 \cdot 271 \cdot 277 \cdot 281 \cdot 283 \cdot 293 \cdot 307 \cdot 311 \cdot 313 \cdot 317 \cdot 331 \cdot 337 \cdot 347 \cdot 349 \cdot 353 \cdot 359 \cdot 367 \cdot 373 \cdot \mathbf{587}$

Gracias-Thanks-dhanyavaad



- Pictures of Botero paintings taken by the author in the Botero museum, Bogotá, Colombia.
- Thanks are due to Jean-Luc Beuchat, Daniel Cervantes-Vázquez and Jesús Chi-Domínguez for designing several of the animations of this presentation
- The AMOR team composed by Gora Adj, Alfred Menezes, Thomaz Oliveira and FRH made several of the contributions presented on the Discrete Logarithm Problem for small characteristic
- The Montgomery ladder material presented in this talk is joint work with Thomaz Oliveira, Julio César López-Hernández, Hüseyin Hisil and Armando Faz-Hernández