Elliptic Curve Cryptography

D. Stebila

School of Mathematical Sciences, QUT

Thursday, August 30, 2012

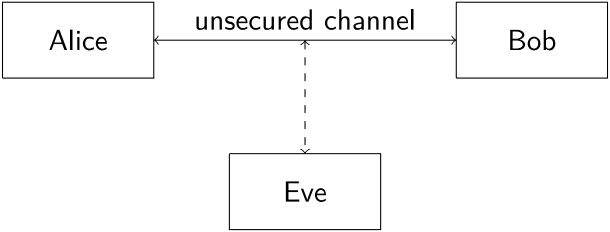
Outline

1. Cryptography
2. Elliptic curves
3. Elliptic curves in practice
4. Elliptic curves in theory
5. Elliptic curves at QUT

Cryptography

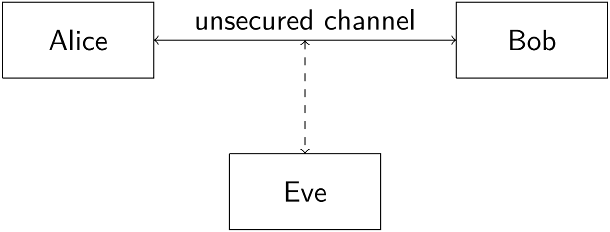
Cryptography

Cryptography aims to provide **confidentiality** and **integrity** of communications.



Cryptography

Cryptography aims to provide **confidentiality** and **integrity** of communications.



- **Symmetric key cryptography:** Alice and Bob share a secret key k that Eve does not know. (Fast!)
- **Public key cryptography:** Alice and Bob have each other's public keys pk_A and pk_B but no shared secrets. (Slow!)

Public key cryptography

Alice generates a pair of related keys:

- pk_A: her **public key**, which she gives to anyone who wants to communicate with her
- sk_A: her **private key**, which she keeps secret

It should be hard for an attacker to compute sk_A only given pk_A.

Public key cryptography

Alice generates a pair of related keys:

- pk_A: her **public key**, which she gives to anyone who wants to communicate with her
- sk_A: her **private key**, which she keeps secret

It should be hard for an attacker to compute sk_A only given pk_A.

Once Alice and Bob get each other's public keys, they can do:

- **public key encryption**: Alice encrypts a message m under Bob's public key pk_B to obtain a ciphertext c; only someone who knows sk_B can decrypt c and recover the message m

Public key cryptography

Alice generates a pair of related keys:

- pk_A: her **public key**, which she gives to anyone who wants to communicate with her
- sk_A: her **private key**, which she keeps secret

It should be hard for an attacker to compute sk_A only given pk_A.

Once Alice and Bob get each other's public keys, they can do:

- **public key encryption**: Alice encrypts a message m under Bob's public key pk_B to obtain a ciphertext c; only someone who knows sk_B can decrypt c and recover the message m
- **digital signatures**: Alice constructs a signature σ for a message m using sk_A; anyone with pk_A can verify whether (m, σ) came from someone who knows sk_A or not

Public key cryptography

Alice generates a pair of related keys:

- pk_A: her **public key**, which she gives to anyone who wants to communicate with her
- sk_A: her **private key**, which she keeps secret

It should be hard for an attacker to compute sk_A only given pk_A.

Once Alice and Bob get each other's public keys, they can do:

- **public key encryption**: Alice encrypts a message m under Bob's public key pk_B to obtain a ciphertext c; only someone who knows sk_B can decrypt c and recover the message m
- **digital signatures**: Alice constructs a signature σ for a message m using sk_A; anyone with pk_A can verify whether (m, σ) came from someone who knows sk_A or not
- **key agreement**: Alice and Bob compute a shared key k that they can use with symmetric encryption

Cryptography on the web

Suppose Alice wants to securely send her credit card number to `bob.com`.

1. Alice obtains a true copy of the public key pk_B for `bob.com`.
2. Alice and Bob run a key agreement protocol to get a shared secret k.
3. Alice and Bob use k with a symmetric cipher to encrypt their communication.

Cryptography on the web

Suppose Alice wants to securely send her credit card number to `bob.com`.

1. Alice obtains a true copy of the public key pk_B for `bob.com`.
2. Alice and Bob run a key agreement protocol to get a shared secret k.
3. Alice and Bob use k with a symmetric cipher to encrypt their communication.

The protocol that implements this is the **Secure Sockets Layer (SSL)** protocol, also known as the **Transport Layer Security (TLS)** protocol, which is the "s" in "`https`".

Modular arithmetic

$a \mod n$

- Let n be a positive integer and a be a non-negative integer.
- $a \mod n$ is the remainder when a is divided by n.
- Example: $12 \mod 5 = 2$

Modular arithmetic

$a \mod n$

- Let n be a positive integer and a be a non-negative integer.
- $a \mod n$ is the remainder when a is divided by n.
- Example: $12 \mod 5 = 2$

primitive root $\bmod n$

- Let g and n be positive integers.
- g is a **primitive root** $\bmod n$ if $g^{n-1} \mod n = 1$ but $g^i \mod n \neq 1$ for any $1 \leq i < n-1$.
- Example:

g	g^2	g^3	g^4	g^5	$g^6 \mod 7$
2	4	$8 = 1$	2	4	1
3	$9 = 2$	6	$18 = 4$	$12 = 5$	$15 = 1$

Diffie–Hellman key exchange (1976)

Goal: Alice and Bob know each other's public keys and want to establish a shared secret key.

Diffie–Hellman key exchange (1976)

Goal: Alice and Bob know each other's public keys and want to establish a shared secret key.

System parameters: p, a large prime number; g, a primitive root $\bmod p$.

Diffie–Hellman key exchange (1976)

Goal: Alice and Bob know each other's public keys and want to establish a shared secret key.

System parameters: p, a large prime number; g, a primitive root $\bmod p$.

$$\begin{array}{lcl}
\underline{\text{Alice}} & & \underline{\text{Bob}} \\
a \leftarrow_R \{2, \ldots, p-1\} & & b \leftarrow_R \{2, \ldots, p-1\} \\
A \leftarrow g^a \mod p & & B \leftarrow g^b \mod p \\
& \xrightarrow{A} & \\
& \xleftarrow{B} & \\
k \leftarrow B^a \mod p & & k' \leftarrow A^b \mod p
\end{array}$$

Diffie–Hellman key exchange (1976)

Goal: Alice and Bob know each other's public keys and want to establish a shared secret key.

System parameters: p, a large prime number; g, a primitive root $\bmod p$.

$$\begin{array}{lcl} \underline{\text{Alice}} & & \underline{\text{Bob}} \\ a \leftarrow_R \{2, \ldots, p-1\} & & b \leftarrow_R \{2, \ldots, p-1\} \\ A \leftarrow g^a \mod p & & B \leftarrow g^b \mod p \\ & \xrightarrow{A} & \\ & \xleftarrow{B} & \\ k \leftarrow B^a \mod p & & k' \leftarrow A^b \mod p \end{array}$$

If Eve does not interfere:

- Alice computes $k = B^a = (g^b)^a = g^{ba} \mod p$
- Bob computes $k' = A^b = (g^a)^b = g^{ab} = g^{ba} \mod p$

Security of Diffie–Hellman key exchange

If Eve can compute the **discrete logarithm** of A to the base $g \pmod p$ then she can find a and compute k.

Security of Diffie–Hellman key exchange

If Eve can compute the **discrete logarithm** of A to the base $g \pmod{p}$ then she can find a and compute k.

1. Is computing discrete logarithms hard?

Security of Diffie–Hellman key exchange

If Eve can compute the **discrete logarithm** of A to the base $g \pmod{p}$ then she can find a and compute k.

1. Is computing discrete logarithms hard?
   - We can't just compute normal logarithms because we are working integers modulo p.

Security of Diffie–Hellman key exchange

If Eve can compute the **discrete logarithm** of A to the base $g \pmod{p}$ then she can find a and compute k.

1. Is computing discrete logarithms hard?
    - We can't just compute normal logarithms because we are working integers modulo p.
    - If p is a very large prime (≥ 1024 bits) and $p - 1$ is divisible by a large prime (≥ 160 bits), then there is no known efficient algorithm.

Security of Diffie–Hellman key exchange

If Eve can compute the **discrete logarithm** of A to the base $g \pmod p$ then she can find a and compute k.

1. Is computing discrete logarithms hard?
    - We can't just compute normal logarithms because we are working integers modulo p.
    - If p is a very large prime (≥ 1024 bits) and $p-1$ is divisible by a large prime (≥ 160 bits), then there is no known efficient algorithm.
    - Still an open problem.

Security of Diffie–Hellman key exchange

If Eve can compute the **discrete logarithm** of A to the base $g \pmod p$ then she can find a and compute k.

1. Is computing discrete logarithms hard?
   - We can't just compute normal logarithms because we are working integers modulo p.
   - If p is a very large prime (≥ 1024 bits) and $p-1$ is divisible by a large prime (≥ 160 bits), then there is no known efficient algorithm.
   - Still an open problem.
   - Caveat: an efficient quantum algorithm is known, but large-scale quantum computers can't be built (yet).

Security of Diffie–Hellman key exchange

If Eve can compute the **discrete logarithm** of A to the base $g \pmod{p}$ then she can find a and compute k.

1. Is computing discrete logarithms hard?
   - We can't just compute normal logarithms because we are working integers modulo p.
   - If p is a very large prime (≥ 1024 bits) and $p - 1$ is divisible by a large prime (≥ 160 bits), then there is no known efficient algorithm.
   - Still an open problem.
   - Caveat: an efficient quantum algorithm is known, but large-scale quantum computers can't be built (yet).
2. Is there any other way of computing k?

Security of Diffie–Hellman key exchange

If Eve can compute the **discrete logarithm** of A to the base $g \pmod p$ then she can find a and compute k.

1. Is computing discrete logarithms hard?
   - We can't just compute normal logarithms because we are working integers modulo p.
   - If p is a very large prime (≥ 1024 bits) and $p - 1$ is divisible by a large prime (≥ 160 bits), then there is no known efficient algorithm.
   - Still an open problem.
   - Caveat: an efficient quantum algorithm is known, but large-scale quantum computers can't be built (yet).
2. Is there any other way of computing k?
   - Not that we know of. But to prove that's the case is an open problem.

Security of Diffie–Hellman key exchange

Let p be a prime and $p-1$ be divisible by a suitably large prime. Then the **best known (classical) algorithm** for computing discrete logarithms takes

$$L_p = \exp\left(\sqrt[3]{\frac{64}{9}}(\ln p)^{1/3}(\ln\ln p)^{2/3}\right)$$

operations.

Security of Diffie–Hellman key exchange

Let p be a prime and $p-1$ be divisible by a suitably large prime. Then the **best known (classical) algorithm** for computing discrete logarithms takes

$$L_p = \exp\left(\sqrt[3]{\frac{64}{9}}(\ln p)^{1/3}(\ln\ln p)^{2/3}\right)$$

operations.

p	L_p	time in years for 10^6 PCs
1024 bits	$2^{86.8}$	$2^{10.5} = 1390$
2048 bits	$2^{116.9}$	$2^{40.6} = 1.6 \times 10^{12}$
4096 bits	$2^{156.5}$	$2^{80.2} = 1.4 \times 10^{24}$

operations per year:

$$10^6 \text{ PCs} \times 365 \text{ days} \times 24 \text{ hrs} \times 60 \text{ mins} \times 60 \text{ secs} \times 3 \times 10^9 \text{ ops} = 2^{76.3}$$

Diffie–Hellman key exchange in a group

- A **group** is a set G along with an operation $\cdot$ which is closed, associative, has an identity element, and inverses exist.
  Example: $\mathbb{Q} \setminus \{0\}$ under multiplication.
- An **abelian group** is a group where the operation is commutative.
- A group has **order** q if there exists an element $g \in G$ such that $\{g^0, g^1, \ldots, g^{q-1}\} = G$; g is called a **generator**

Diffie–Hellman key exchange in a group

- A **group** is a set G along with an operation $\cdot$ which is closed, associative, has an identity element, and inverses exist.
  Example: $\mathbb{Q} \setminus \{0\}$ under multiplication.
- An **abelian group** is a group where the operation is commutative.
- A group has **order** q if there exists an element $g \in G$ such that $\{g^0, g^1, \ldots, g^{q-1}\} = G$; g is called a **generator**

System parameters: a group G with a generator g of large prime order q

Diffie–Hellman key exchange in a group

- A **group** is a set G along with an operation $\cdot$ which is closed, associative, has an identity element, and inverses exist.
  Example: $\mathbb{Q} \setminus \{0\}$ under multiplication.
- An **abelian group** is a group where the operation is commutative.
- A group has **order** q if there exists an element $g \in G$ such that $\{g^0, g^1, \ldots, g^{q-1}\} = G$; g is called a **generator**

System parameters: a group G with a generator g of large prime order q

<u>Alice</u>		<u>Bob</u>
$a \leftarrow_R \{2, \ldots, q-1\}$		$b \leftarrow_R \{2, \ldots, q-1\}$
$A \leftarrow g^a$		$B \leftarrow g^b$
	$\xrightarrow{A}$	
	$\xleftarrow{B}$	
$k \leftarrow B^a (= g^{ba})$		$k' \leftarrow A^b (= g^{ab})$

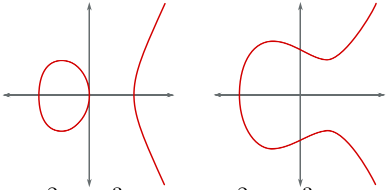
Elliptic curves

Elliptic curves

An **elliptic curve over $\mathbb{R}$** is the set of real points satisfying an equation of the form

$$y^2 = x^3 + ax + b$$

where $a, b \in \mathbb{R}$ and $4a^3 + 27b^2 \neq 0$.



$y^2 = x^3 - x$ $\quad$ $y^2 = x^3 - x + 1$

Elliptic curve points as a group

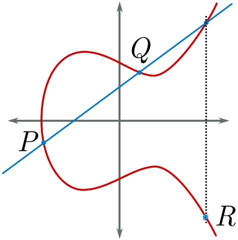
We will construct a group consisting of the points of an elliptic curve under the operation of **point addition**.

Elliptic curve points as a group

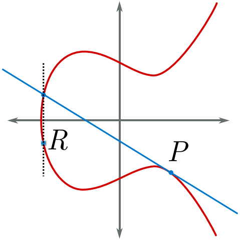
We will construct a group consisting of the points of an elliptic curve under the operation of **point addition**. Define a "point at infinity O".

Elliptic curve points as a group

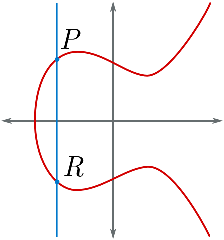
We will construct a group consisting of the points of an elliptic curve under the operation of **point addition**. Define a “point at infinity O”.



$P + Q = R$



$2P = R$



$P + R = O$

From the geometric intuition, we can easily compute algebraic formulas for point addition, point doubling, and point negation.

Elliptic curve scalar–point multiplication

Having defined point addition and point doubling, we can define **scalar–point multiplication**:

$$kP = \underbrace{P + P + \cdots + P}_{k}$$

Elliptic curve scalar–point multiplication

Having defined point addition and point doubling, we can define **scalar–point multiplication**:

$$kP = \underbrace{P + P + \cdots + P}_{k}$$

We can compute kP more efficiently using the **double-and-add** algorithm:

$$5P = 2(2(P)) + P$$

Elliptic curve scalar–point multiplication

Having defined point addition and point doubling, we can define **scalar–point multiplication**:

$$kP = \underbrace{P + P + \cdots + P}_{k}$$

We can compute kP more efficiently using the **double-and-add** algorithm:

$$5P = 2(2(P)) + P$$

Input: $k = (k_{\ell-1}, \ldots, k_1, k_0)_2,\ P$

1. $Q \leftarrow O$
2. for i from $\ell - 1$ to 0 do:
   - 2.1 $Q \leftarrow 2Q$
   - 2.2 if $k_i = 1$ then $Q \leftarrow Q + P$

Output: $Q = kP$

Elliptic curves over prime fields

Let p be a prime. An **elliptic curve over $\mathbb{Z}_p$** is the set of integer points $\mod p$ satisfying an equation of the form

$$y^2 = x^3 + ax + b \mod p$$

where $a, b \in \mathbb{Z}_p$ and $4a^3 + 27b^2 \neq 0 \mod p$.

Elliptic curve Diffie–Hellman key exchange

System parameters: a prime p, an elliptic curve $y^2 = x^3 + ax + b$, and a point P which is a generator of group of prime order q

Elliptic curve Diffie–Hellman key exchange

System parameters: a prime p, an elliptic curve $y^2 = x^3 + ax + b$, and a point P which is a generator of group of prime order q

$$
\begin{array}{lcl}
\underline{\text{Alice}} & & \underline{\text{Bob}} \\
a \leftarrow_R \{2, \ldots, q-1\} & & b \leftarrow_R \{2, \ldots, q-1\} \\
A \leftarrow aP & & B \leftarrow bP \\
 & \xrightarrow{A} & \\
 & \xleftarrow{B} & \\
k \leftarrow aB (= abP) & & k' \leftarrow bA (= baP)
\end{array}
$$

Security of ECDH key exchange

If Eve can compute **elliptic curve discrete logarithms**, then she can find a and compute k.

The best known (classical) algorithm for computing elliptic curve discrete logarithms takes about $\sqrt{q}$ operations.

Security of ECDH key exchange

If Eve can compute **elliptic curve discrete logarithms**, then she can find a and compute k.

The best known (classical) algorithm for computing elliptic curve discrete logarithms takes about $\sqrt{q}$ operations.

DH $\bmod p$		ECDH		
p	L_p	q	$\sqrt{q}$	time in years for 10^6 PCs
1024 bits	$2^{86.8}$	174 bits	2^{87}	$2^{10.5} = 1390$
2048 bits	$2^{116.9}$	235 bits	2^{117}	$2^{40.6} = 1.6 \times 10^{12}$
4096 bits	$2^{156.5}$	321 bits	2^{157}	$2^{80.2} = 1.4 \times 10^{24}$

Security of ECDH key exchange

If Eve can compute **elliptic curve discrete logarithms**, then she can find a and compute k.

The best known (classical) algorithm for computing elliptic curve discrete logarithms takes about $\sqrt{q}$ operations.

DH $\bmod p$		ECDH		
p	L_p	q	$\sqrt{q}$	time in years for 10^6 PCs
1024 bits	$2^{86.8}$	174 bits	2^{87}	$2^{10.5} = 1390$
2048 bits	$2^{116.9}$	235 bits	2^{117}	$2^{40.6} = 1.6 \times 10^{12}$
4096 bits	$2^{156.5}$	321 bits	2^{157}	$2^{80.2} = 1.4 \times 10^{24}$

ECDH can achieve the **same level of security** with much **smaller values**.
Smaller values $\implies$ faster computation.

Elliptic curves in practice

ECC on the Internet

Most modern major web browsers and web servers support ECC:

- Microsoft Internet Explorer and Internet Information Server
- Mozilla Firefox**
- Google Chrome*
- Apache**

ECC on the Internet

Most modern major web browsers and web servers support ECC:

- Microsoft Internet Explorer and Internet Information Server
- Mozilla Firefox**
- Google Chrome*
- Apache**

Use of ECC is not too widespread, yet. But in November 2011, **Google** changed their configuration so that all their web servers would use **ECDH** as their default ciphersuite.

- Faster computation.

ECC on the Internet

Most modern major web browsers and web servers support ECC:

- Microsoft Internet Explorer and Internet Information Server
- Mozilla Firefox**
- Google Chrome*
- Apache**

Use of ECC is not too widespread, yet. But in November 2011, **Google** changed their configuration so that all their web servers would use **ECDH** as their default ciphersuite.

- Faster computation.
- Better security compared to existing RSA ciphersuites.

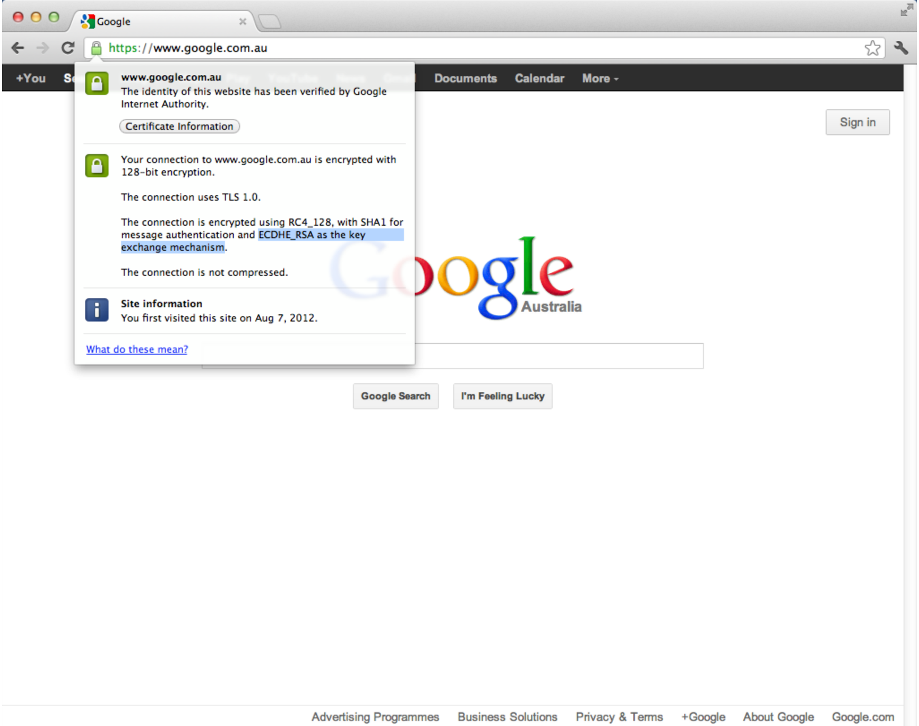
ECC on the Internet

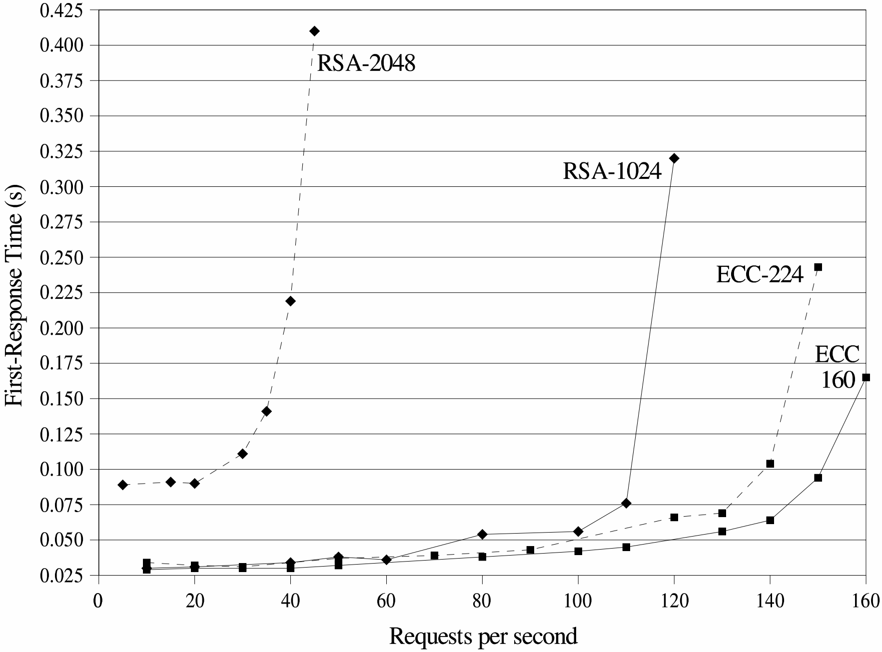
Most modern major web browsers and web servers support ECC:

- Microsoft Internet Explorer and Internet Information Server
- Mozilla Firefox**
- Google Chrome*
- Apache**

Use of ECC is not too widespread, yet. But in November 2011, **Google** changed their configuration so that all their web servers would use **ECDH** as their default ciphersuite.

- Faster computation.
- Better security compared to existing RSA ciphersuites.
- **Forward security**: If Google's long term public key gets compromised later, your current encryptions can't be broken.


Google
https://www.google.com.au
+You
Documents
Calendar
More
www.google.com.au
The identity of this website has been verified by Google Internet Authority.
Certificate Information
Your connection to www.google.com.au is encrypted with 128-bit encryption.
The connection uses TLS 1.0.
The connection is encrypted using RC4_128, with SHA1 for message authentication and ECDHE_RSA as the key exchange mechanism.
The connection is not compressed.
Site information
You first visited this site on Aug 7, 2012.
What do these mean?
Sign in
Google
Australia
Google Search
I'm Feeling Lucky
Advertising Programmes
Business Solutions
Privacy & Terms
+Google
About Google
Google.com



First-Response Time (s)
0.425
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0
20
40
60
80
100
120
140
160
Requests per second
RSA-2048
RSA-1024
ECC-224
ECC
160

Side-channel attacks on point multiplication

- The basic **double-and-add** point multiplication algorithm does an extra operation whenever the key bit is 1.
- If an adversary can see when your computer does that extra operation, she can recover your key.

Side-channel attacks on point multiplication

- The basic **double-and-add** point multiplication algorithm does an extra operation whenever the key bit is 1.
- If an adversary can see when your computer does that extra operation, she can recover your key.
- How? **Side-channels** such as electricity usage, radiation, or timing.

Side-channel attacks on point multiplication

- The basic **double-and-add** point multiplication algorithm does an extra operation whenever the key bit is 1.
- If an adversary can see when your computer does that extra operation, she can recover your key.
- How? **Side-channels** such as electricity usage, radiation, or timing.

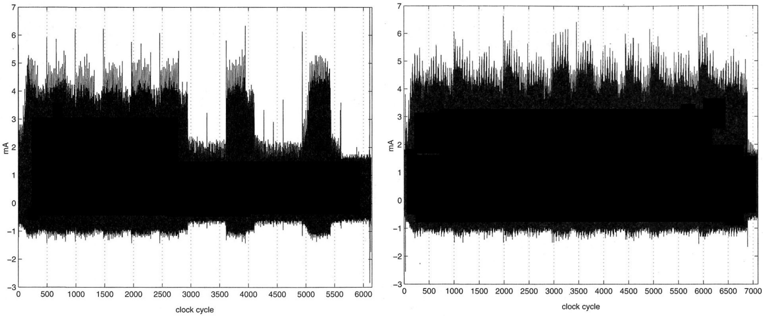


Figure : Point doubling and point addition

Side-channel attacks on point multiplication

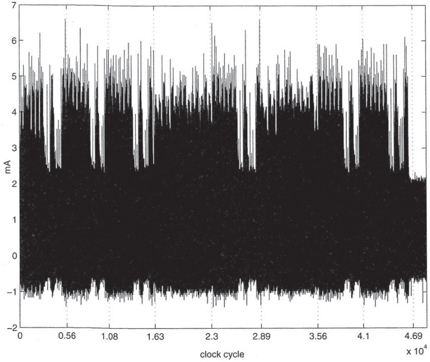


Figure : Point multiplication

Side-channel attacks on point multiplication

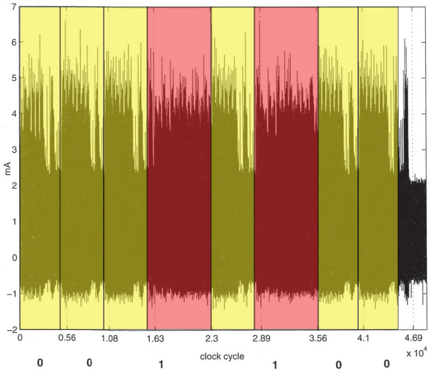


Figure : Point multiplication with additions and doublings identified

Elliptic curves in theory

Elliptic curve pairings

A **bilinear pairing** is a function e that given g^a and g^b can compute

$$e(g^a, g^b) = e(g, g)^{ab}$$

Elliptic curve pairings

A **bilinear pairing** is a function e that given g^a and g^b can compute

$$e(g^a, g^b) = e(g, g)^{ab}$$

Pairings can be used to construct many cryptographic protocols:

- **3-party Diffie–Hellman key exchange**:
  Alice $A = g^a$, Bob $B = g^b$, Charlie $C = g^c$
  $k = e(g, g)^{abc} = e(B, C)^a = e(A, C)^b = e(A, B)^c$

Elliptic curve pairings

A **bilinear pairing** is a function e that given g^a and g^b can compute

$$e(g^a, g^b) = e(g, g)^{ab}$$

Pairings can be used to construct many cryptographic protocols:

- **3-party Diffie–Hellman key exchange**:
  Alice $A = g^a$, Bob $B = g^b$, Charlie $C = g^c$
  $k = e(g, g)^{abc} = e(B, C)^a = e(A, C)^b = e(A, B)^c$
- **identity-based encryption**:
  Instead of having to get Bob's public key, Alice can encrypt a message based on Bob's identity, such as `bob@gmail.com`.

Fermat's Last Theorem

- **Theorem (Fermat, 1647).** There exist no positive integers a, b, c that satisfy the equation

$$a^n + b^n = c^n$$

for any integer $n > 2$.

Fermat's Last Theorem

- **Theorem (Fermat, 1647).** There exist no positive integers a, b, c that satisfy the equation

$$a^n + b^n = c^n$$

  for any integer $n > 2$.
- *Proof.* (1637–1994) "I have discovered a truly marvellous proof of this, which this margin is too narrow to contain."

Fermat's Last Theorem

- **Theorem (Fermat, 1647).** There exist no positive integers a, b, c that satisfy the equation

$$a^n + b^n = c^n$$

  for any integer $n > 2$.
- *Proof.* (1637–1994) "I have discovered a truly marvellous proof of this, which this margin is too narrow to contain."
- **Frey (1984).** If Fermat's equation had a solution (a, b, c) for $p > 2$, then the elliptic curve

$$y^2 = x(x - a^p)(x - b^p)$$

  would have unusual properties (violate the modularity theorem).

Fermat's Last Theorem

- **Theorem (Fermat, 1647).** There exist no positive integers a, b, c that satisfy the equation

$$a^n + b^n = c^n$$

  for any integer $n > 2$.
- *Proof.* (1637–1994) "I have discovered a truly marvellous proof of this, which this margin is too narrow to contain."
- **Frey (1984).** If Fermat's equation had a solution (a, b, c) for $p > 2$, then the elliptic curve

$$y^2 = x(x - a^p)(x - b^p)$$

  would have unusual properties (violate the modularity theorem).
- **Wiles (1995).** Proof of modularity theorem and Fermat's Last Theorem. 100+ pages.

Elliptic curve cryptography at QUT

Elliptic curve cryptography at QUT

Research:

- early implementations of ECC
- fast algorithms for ECC and pairings
- side-channel-resistant algorithms for ECC
- use of ECC and pairings in designing new cryptographic schemes

Elliptic curve cryptography at QUT

Research:

- early implementations of ECC
- fast algorithms for ECC and pairings
- side-channel-resistant algorithms for ECC
- use of ECC and pairings in designing new cryptographic schemes

Teaching:

- **MAB461 Discrete Mathematics**:
  modular arithmetic, number theory, RSA public key cryptography
- **MAN778 Applications of Discrete Mathematics**:
  advanced number theory, group theory, Diffie–Hellman, introduction to elliptic curves, provable security
- **INN355 Cryptology and Protocols**:
  symmetric and public key cryptography
- **INN652 Advanced Cryptology**: elliptic curve cryptography