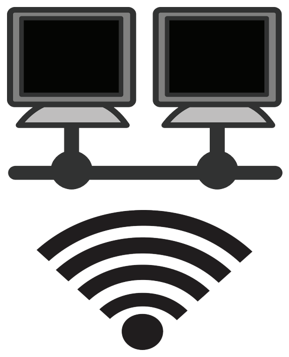
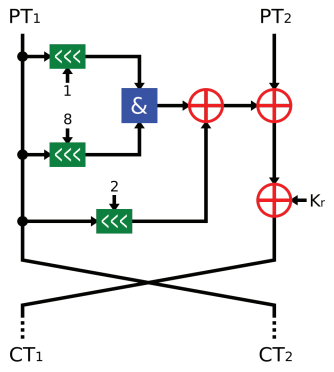
CSE 40567 / 60567: Computer Security





Cryptography 5

Homework #2 has been released. It is due **Tonight** at 11:59PM

See **Assignments Page** on the course website for details

Elliptic curve cryptography

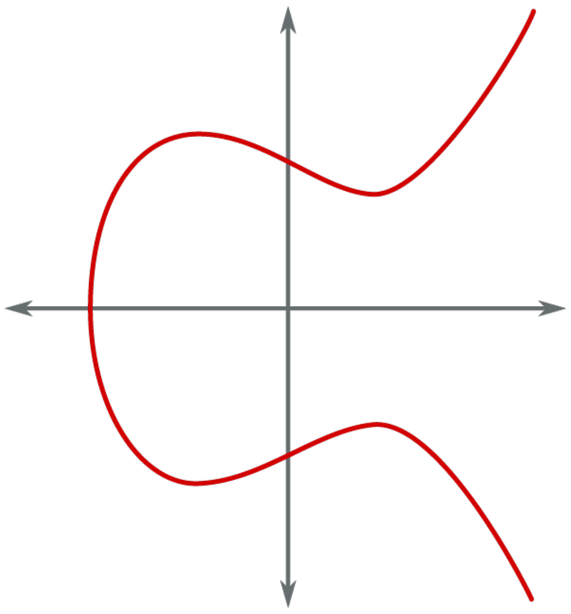
- Next generation of public key cryptography
- Provides a significantly more secure foundation than first generation systems like RSA
- Trouble with RSA: As factoring prime numbers becomes more efficient, key sizes have to grow
    - This slows down the algorithm

http://arstechnica.com/security/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/

Alternative Trapdoor Function

Ideal property: f and f^{-1} become more difficult at the same rate w.r.t. to the size of the numbers at hand

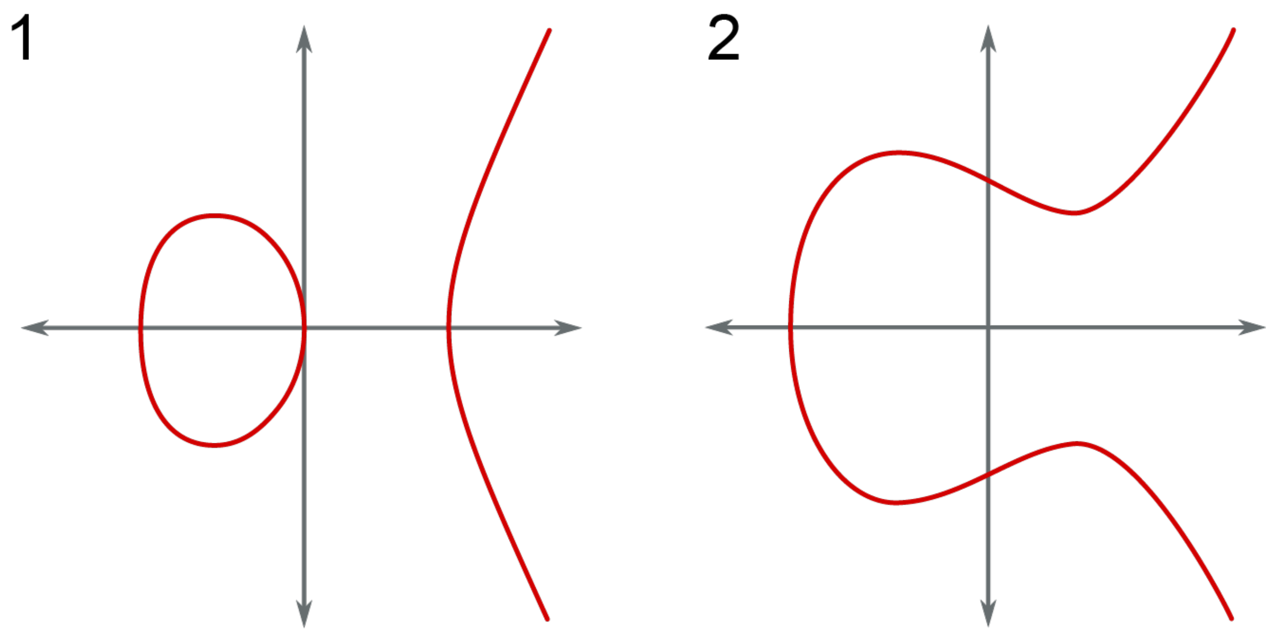
- *Elliptic Curves* satisfy this property
- Unlike factoring, the math isn't as straightforward conceptually



Examples of elliptic curves ⓒ BY-SA 3.0 SuperManu

Elliptic Curves

Definition: set of points satisfying an equation in two variables with degree two in one of the variables and three in the other



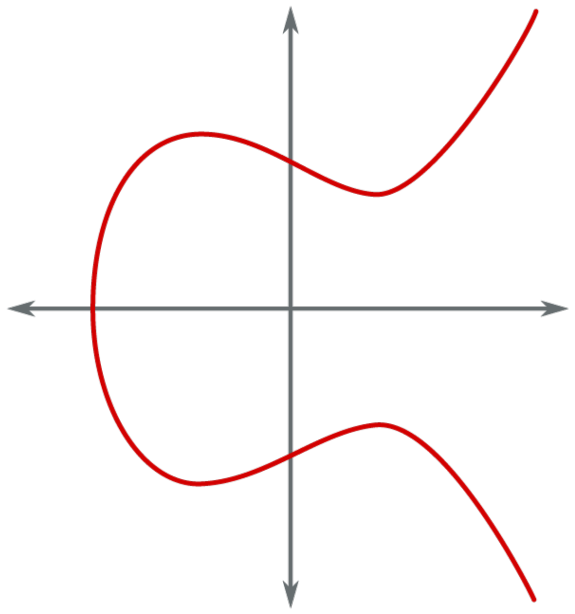
$y^2 = x^3 - x$ $\quad$ $y^2 = x^3 - x + 1$

Examples of elliptic curves (cc) BY-SA 3.0 SuperManu

Symmetry Property

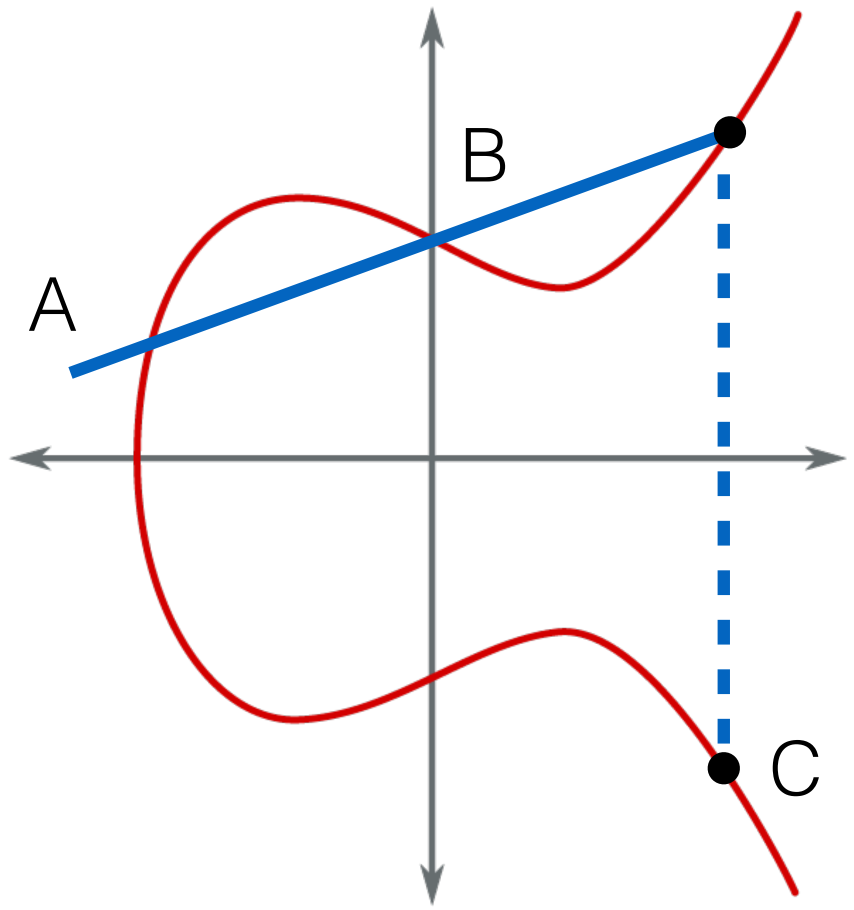
Horizontal Symmetry

- Any point can be reflected over the x-axis and remain on the same curve
- Any non-vertical line will intersect the curve in at most three places



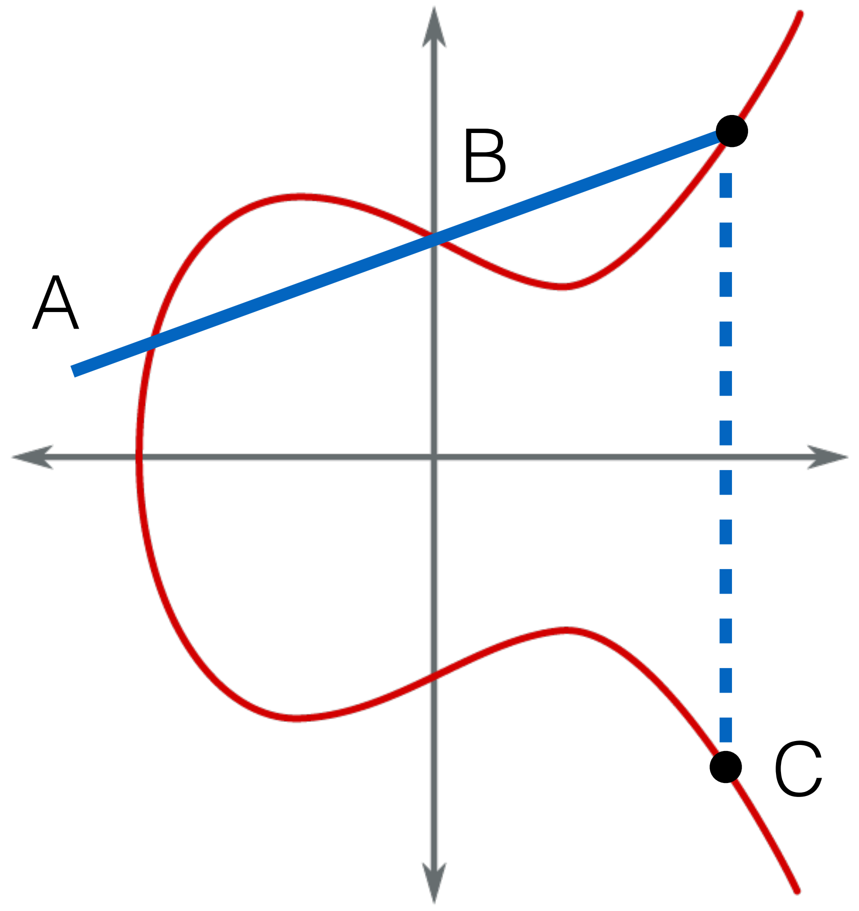
Examples of elliptic curves (cc) BY-SA 3.0 SuperManu

Operations over the curve



Examples of elliptic curves 🅭 BY-SA 3.0 SuperManu

Operations over the curve



Examples of elliptic curves (cc) BY-SA 3.0 SuperManu

Let's call this operation ●

Any two points on the curve can be processed by this operation to get a new point:

A ● B = C

It is possible to apply ● to the same point and subsequent results n number of times

A ● A = B

A ● B = C

A ● C = D

Elliptic Curve Trapdoor

Assume there is a starting point and an ending point

The starting point goes through n ● operations to arrive at the ending point

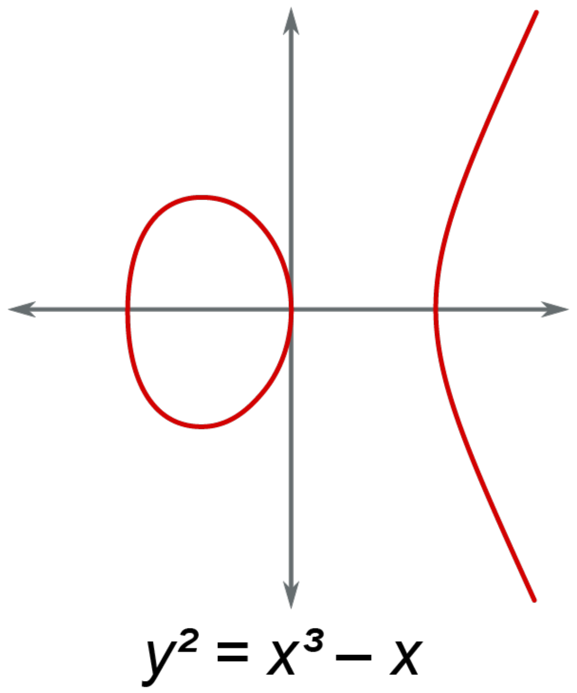
Finding out n when only the starting point and ending point are known is a hard problem

Elliptic Curves for Cryptography

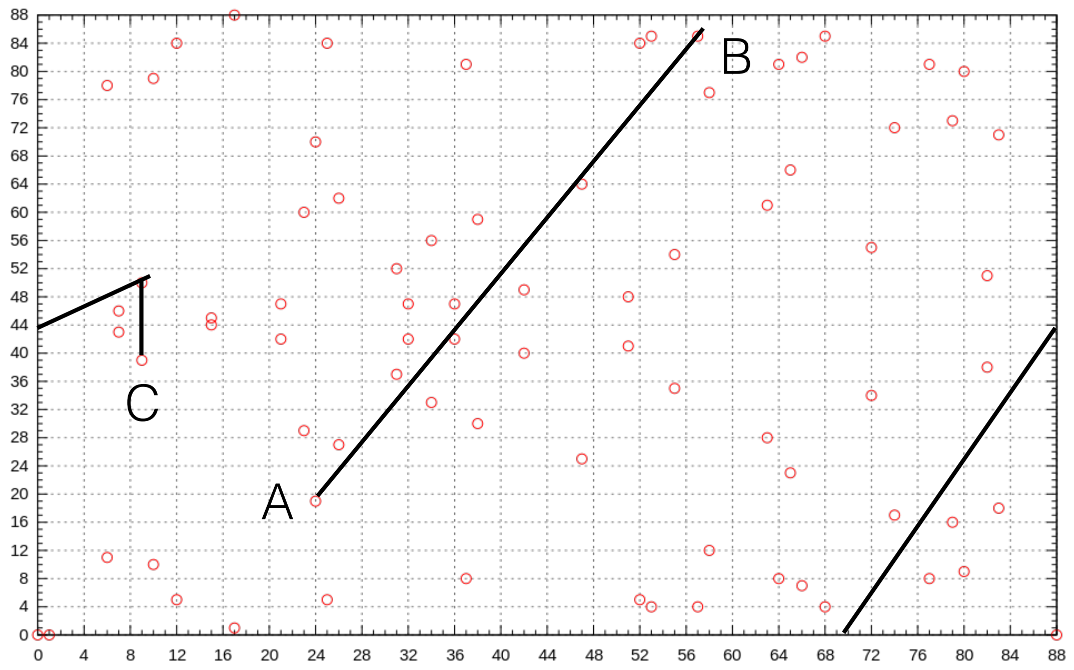
Numbers need to be restricted to a fixed range

- Points on the curve are whole numbers in a fixed range.
- Numbers roll over when a maximum is hit.
- The maximum is chosen to be a prime number. In this case, the curve is called a prime curve and is suitable for cryptography.

Elliptic Curves over Finite Fields



Set of affine points of elliptic curve $y^2 = x^3 - x$ over finite field $\mathbf{F}89$.



Messages are represented as points on the curve

Elliptic curve cryptosystem

- Choose the following elements:
  - a prime number as a maximum
  - a curve equation,
  - and a public point on the curve.
- A private key is a number n, and a public key is the public point "dotted" with itself n times.
- Computing the private key from the public key is called the elliptic curve discrete logarithm function.

$\nearrow$

Trapdoor Function

Security of Elliptic Curve Cryptography

- Compared to RSA, high security with shorter keys and excellent computational performance
- For primitives of the same size, solving elliptic curve discrete logarithms is a much harder problem than prime number factorization
    - Elliptic Curve algorithms are thus more secure than RSA

Global Security Model

Let's make the security comparison a bit more tangible:

The amount of energy needed to break a cryptographic algorithm can be expressed as how much water the energy can boil

228-bit RSA key: less energy than it takes to boil a teaspoon of water

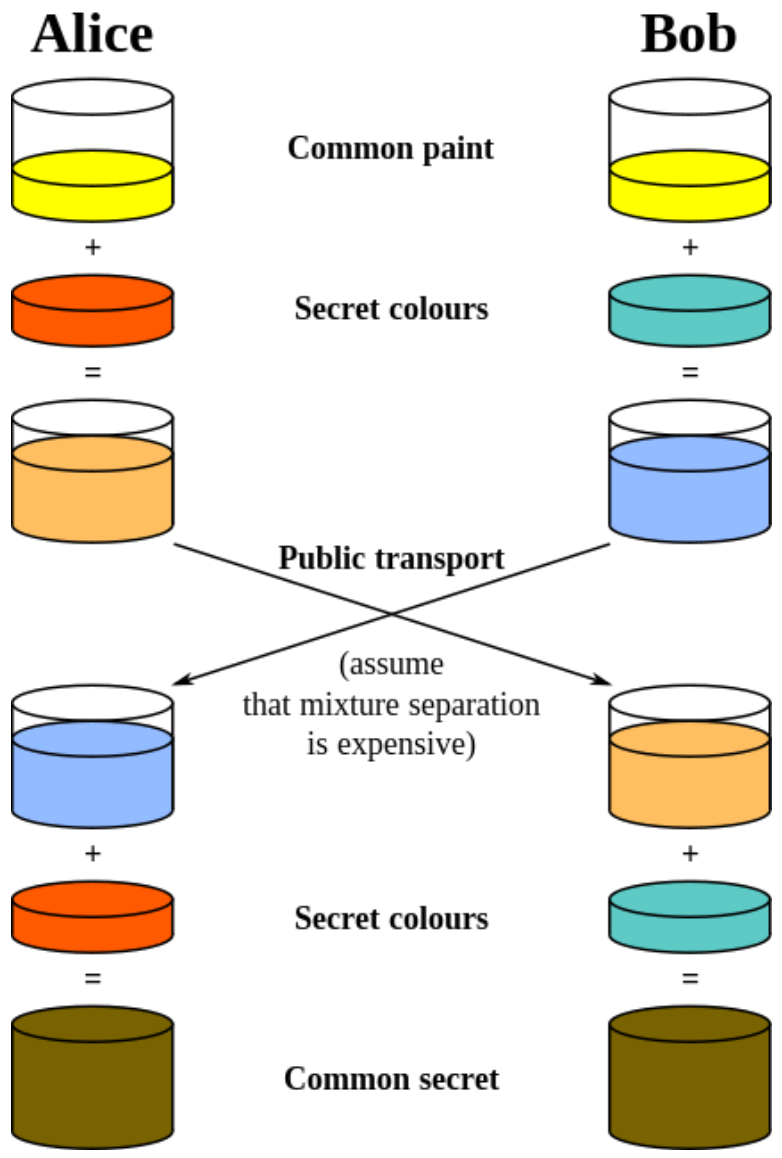
228-bit EC key: enough energy to boil all of the water on earth

Lenstra et al., "Universal security: from bits and mips to pools, lakes – and beyond," in *Number Theory and Cryptography*, 2013.

What is the relationship to Diffie-Hellman key exchange?

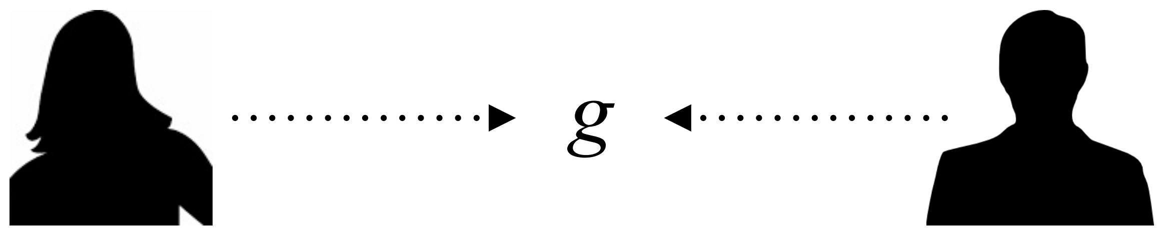
Like RSA, EC is commonly used to transmit a key to Alice and Bob, which will be used with a symmetric key cipher.

Alternative protocol to RSA: Diffie-Hellman



Elliptic Curve Diffie-Hellman

1. Alice and Bob agree on a shared curve generation parameter g



2. Alice generates her private key a and calculates her public key A

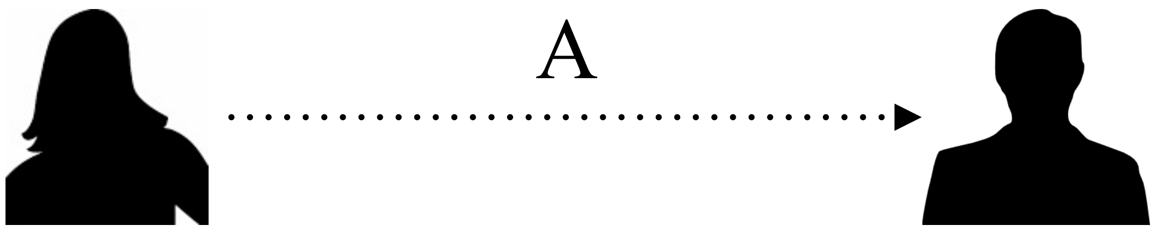
$$\mathrm{A} = a \times g$$

Elliptic Curve Diffie-Hellman

3. Bob generates his private key b and calculates his public key B

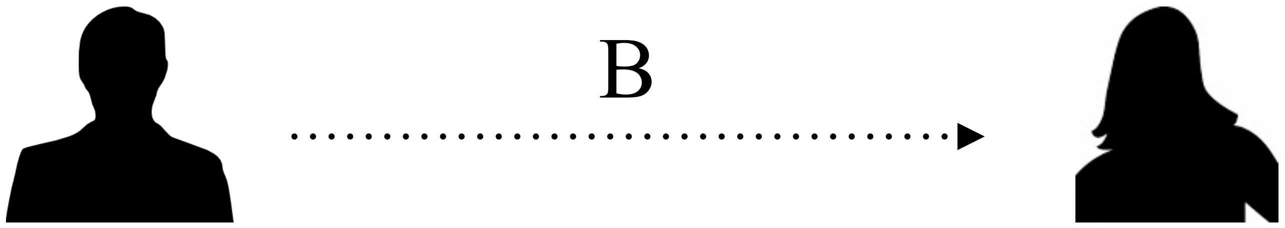
$$\mathrm{B} = b \times g$$

4. Alice sends her public key to Bob



Elliptic Curve Diffie-Hellman

5. Bob sends his public key to Alice



6. Alice calculates the shared secret

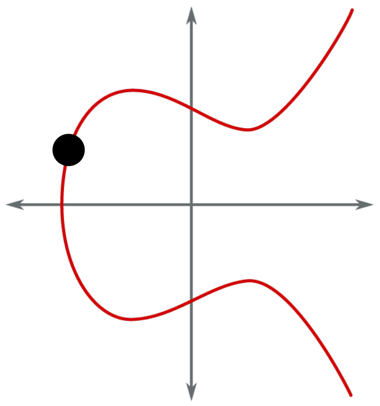
$$B \times a = a \times b \times g$$

Elliptic Curve Diffie-Hellman

7. Bob calculates the shared secret

$$\mathrm{A} \times b = a \times b \times g$$

8. The shared secret is the x coordinate of the calculated point on the curve



Examples of elliptic curves (cc) BY-SA 3.0 SuperManu

Choosing parameters

- Choice of modulus (public key): 256 bits is suitable for 128-bit ciphers
- Trouble is the choice of curve
  - Set of curves standardized by NIST [1]
    - Snowden disclosures: NSA tampered with the NIST process for other crypto designs
  - Brainpool curves [2]
    - May also have been manipulated
  - Bernstein's Curve25519 [3]

1. NIST FIPS 186-4
2. http://safecurves.cr.yp.to/bada55/bada55-20140722.pdf
3. http://cr.yp.to/ecdh/curve25519-20060209.pdf

Curve25519

$$y^2 = x^3 + 486662x^2 + x$$

Prime field: $2^{255} - 19$ Base point: $x = 9$

Advantages:

- One of the fastest known curves
- Not covered by any patents
- Less susceptible to weak random-number generators (making side-channel attacks more difficult)
- Not tainted by suspicious standards process

Avoidable Pitfalls of EC

Choosing weak primitives

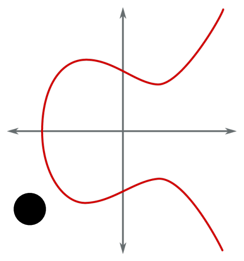
Use a modulus (public key) that is at least 256 bits

Choosing an unsafe curve

Consult http://safecurves.cr.yp.to/ before choosing a curve

Choosing an implementation that is vulnerable to invalid curve attacks

Use Curve25519 (via NaCL)



Examples of elliptic curves (cc) BY-SA 3.0 SuperManu

NaCl: Networking and Cryptography library (https://nacl.cr.yp.to)

Library designed to avoid programming mistakes endemic to low-level libraries

D. J. Bernstein's implementation of Curve25519, plus:

- No data flow from secrets to load addresses
- No data flow from secrets to branch conditions
- No padding oracles
- Centralizing randomness
- Avoiding unnecessary randomness
- Extremely high speed

EC functions in LibreSSL / OpenSSL

```
#include <openssl/ec.h>

EC_KEY *EC_KEY_new_by_curve_name(int nid)
void EC_KEY_free(EC_KEY *key);

int EC_KEY_set_private_key(EC_KEY *key, const BIGNUM *prv);
int EC_KEY_set_public_key(EC_KEY *key, const EC_POINT *pub);
const EC_POINT *EC_KEY_get0_public_key(const EC_KEY *key);

int EC_GROUP_get_degree(const EC_GROUP *group);
int ECDH_compute_key(void *out, size_t outlen, const
   EC_POINT *pub_key, EC_KEY *ecdh, void *(*KDF)(const void
   *in, size_t inlen, void *out, size_t *outlen));

int EC_KEY_generate_key(EC_KEY *key);
```

Digital Signatures

Handwritten Signatures

Sometimes we need to verify information in a protocol

B Franklin

Signatures have long been used for proof of authorship, and have not gone away in this digital age.

What's good and bad about them?

What’s good about signatures?

1. The signature is authentic.
2. The signature is unforgettable.
3. The signature is not reusable.
4. The signed document is unalterable.
5. The signature cannot be repudiated.

None of these statements is completely true, but we’ll live with the shortcomings for now…

Digital Signatures

We'd like to facilitate signatures on computers, but there are two fundamental problems:

1. **Computer files are trivial to copy**
2. **Computer files are easy to modify after they are signed**

These problems can be addressed with cryptography

Signing documents with public key cryptography

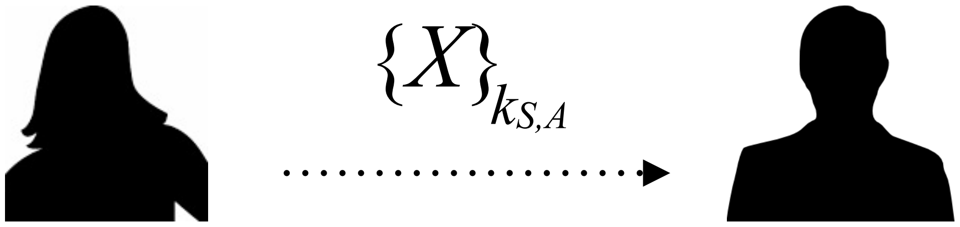
Basic Protocol:

1. Alice encrypts the document with her private key, thus signing it

2. Alice sends the signed document to Bob

3. Bob decrypts the document with Alice's public key, thus verifying the signature

$\{X\}_{k_{S,A}}$



$\{X\}_{k_{S,A}}$

$\{X\}_{k_{S,A}}, k_{P,A} = X$

The protocol satisfies the signature requirements

1. The signature is authentic. Bob verifies the message with Alice’s public key.

2. The signature is unforgettable. Only Alice knows her private key.

3. The signature is not reusable. The signature (ciphertext) is a function of the document.

4. The signed document is unalterable. Any modification to the document means it can’t be verified with Alice’s public key.

5. The signature cannot be repudiated. Bob doesn’t need Alice’s help to verify her signature.

Signing documents with public key cryptography and one-way hash functions

Public key algorithms are too slow to sign long documents, here's an alternative:

1. Alice produces a one-way hash of a document

$$f(X) = h_X \longleftarrow$$

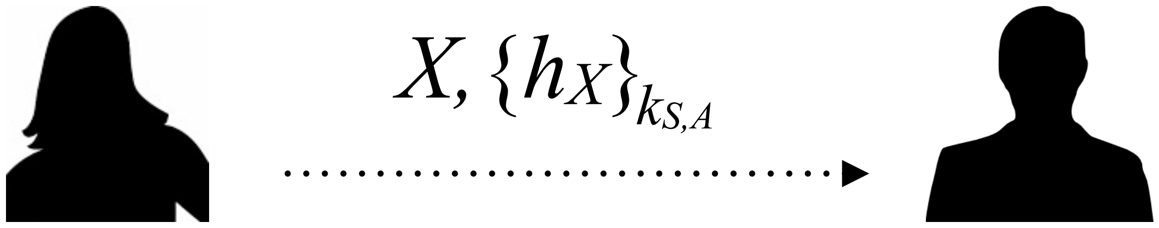
Only 256-bits if using SHA-256

2. Alice encrypts the hash with her private key, thereby signing the document

$$\{h_X\}_{k_{S,A}}$$

Signing documents with public key cryptography and one-way hash functions

3. Alice sends the document and the signed hash to Bob



4. Bob checks the signature

4.1 Hash the document: $f(X) = h_{X,B}$

4.2 Decrypt the signed hash: $\{h_X\}_{k_{S,A}}, k_{P,A} = h_{X,A}$

4.3 Check if hashes match: $h_{X,A} = h_{X,B}$?

Multiple signatures

Easy to support with one-way hash functions. Assume that a hash of document X, h_X, has been generated:

1. Alice signs the hash of the document

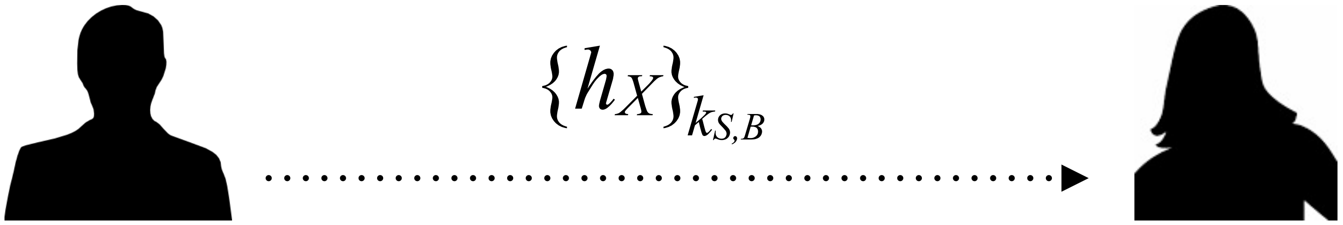
$$\{h_X\}_{k_{S,A}}$$

2. Bob signs the hash of the document

$$\{h_X\}_{k_{S,B}}$$

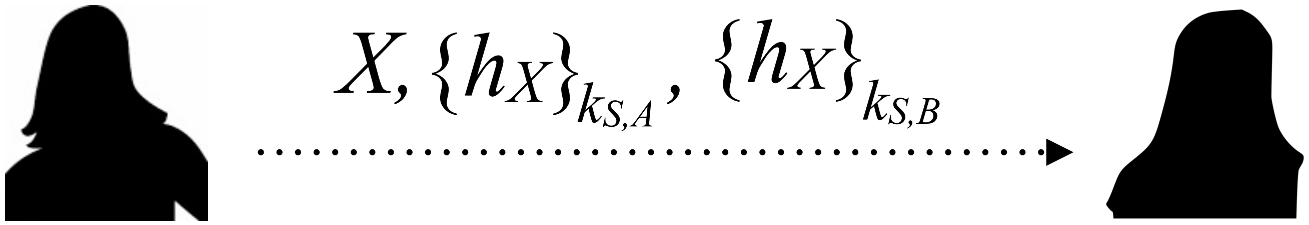
Multiple signatures

3. Bob sends his signature to Alice



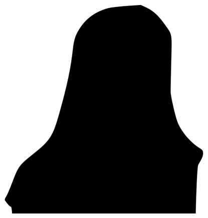
$$\{h_X\}_{k_{S,B}}$$

4. Alice sends the message and signatures to Carol



$$X, \{h_X\}_{k_{S,A}}, \{h_X\}_{k_{S,B}}$$

5. Carol verifies the signatures



$$h_{X,C} = h_{X,A} = h_{X,B}?$$

Non-repudiation and digital signatures

Alice can cheat with digital signatures by signing a document and later claiming that she didn't

1. She signs the document as usual
2. She then anonymously publishes her private key, or conveniently loses it in a public place
3. All documents signed by Alice are repudiated

Possible solution: timestamps

- However, Alice can still claim the key was compromised earlier, or time her signings well

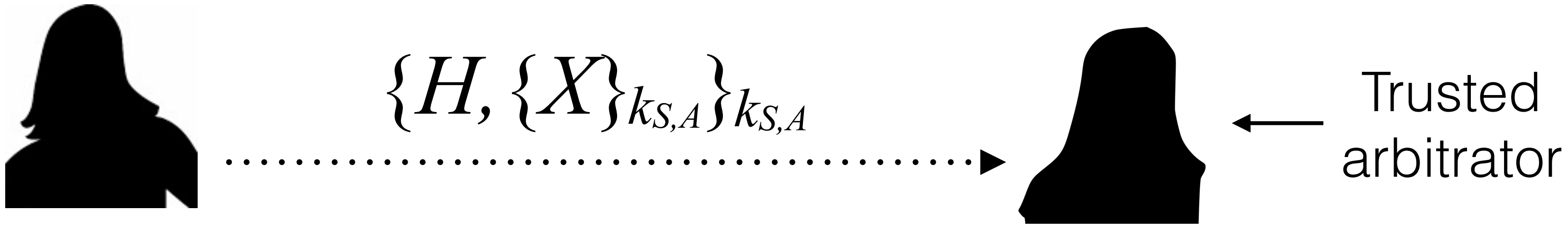
Non-repudiation and digital signatures: protocol

It is possible to design a protocol to guarantee that old signatures are not invalidated by actions taken in disputing new ones:

1. Alice signs a message

$$\{X\}_{k_{S,A}}$$

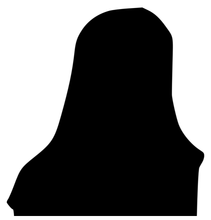
2. Alice generates a header H containing some identifying info., concatenates it with the signed message, signs that, and sends to Carol



$$\{H, \{X\}_{k_{S,A}}\}_{k_{S,A}}$$

Non-repudiation and digital signatures: protocol

3. Carol verifies Alice’s information, adds a timestamp, and signs the updated message

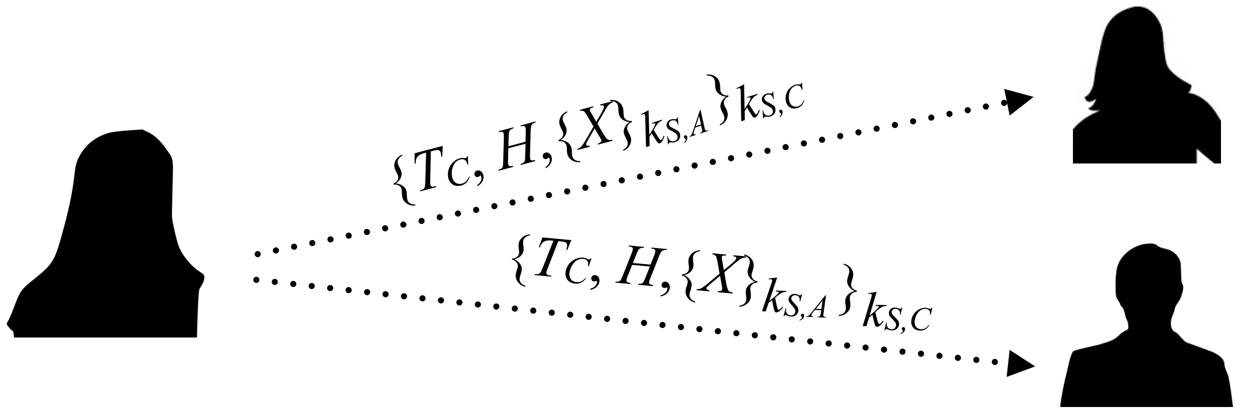


3.1 $\{H,\{X\}_{k_{S,A}}\}_{k_{S,A}}, k_{P,A} = H,\{X\}_{k_{S,A}}$

3.2 $T_C, H,\{X\}_{k_{S,A}}$

3.3 $\{T_C, H,\{X\}_{k_{S,A}}\}_{k_{S,C}}$

4. Carol sends the updated message to Alice and Bob


$\{T_C, H,\{X\}_{k_{S,A}}\}_{k_{S,C}}$
$\{T_C, H,\{X\}_{k_{S,A}}\}_{k_{S,C}}$

Non-repudiation and digital signatures: protocol

5. Bob verifies Carol’s signature, the identifying information, and Alice’s signature

5.1 $\{T_C, H, \{X\}_{k_{S,A}}\}_{k_{S,C}}, k_{P,C} = T_C, H, \{X\}_{k_{S,A}}$

5.2 $\{X\}_{k_{S,A}}, k_{P,A} = X$

6. Alice verifies Carol’s signature, if she did not originate the message, she raises an alert

6.1 $\{T_C, H, \{X\}_{k_{S,A}}\}_{k_{S,C}}, k_{P,C} = T_C, H, \{X\}_{k_{S,A}}$

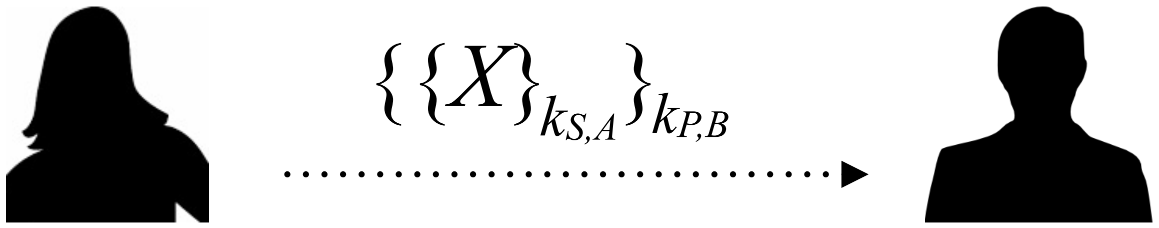
Digital signatures with encryption

Think of the signatures as proof of authorship and the encryption as the envelope

1. Alice signs a message

$\{X\}_{k_{S,A}}$

2. Alice encrypts the message with Bob's public key and sends it to him



$\{\{X\}_{k_{S,A}}\}_{k_{P,B}}$

Digital signatures with encryption

3. Bob decrypts the message with his private key

$$\{\{X\}_{k_{S,A}}\}_{k_{P,B}}, k_{S,B} = \{X\}_{k_{S,A}}$$

4. Bob verifies with Alice's public key and recovers the message

$$\{X\}_{k_{S,A}}, k_{P,A} = X$$

Practical considerations

Alice can use two key pairs

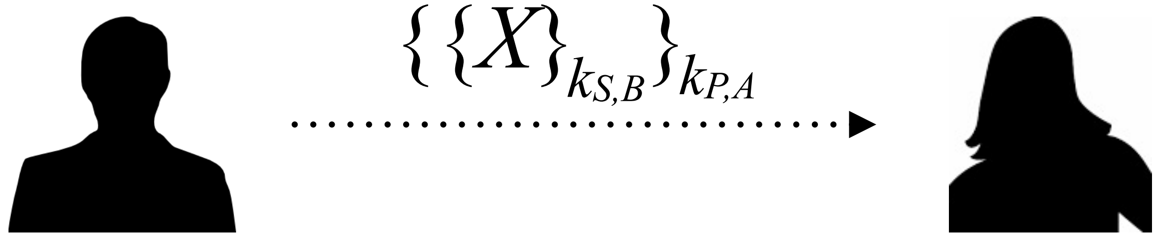
- For encryption: $k_{P,A,Enc}$, $k_{S,A,Enc}$
- For signing: $k_{P,A,Sign}$, $k_{S,A,Sign}$

Advantages: one key pair can be surrendered without affecting the other, one key could be *escrowed* without affecting the other, keys can expire at different times

Timestamps used with the protocol can prevent message reuse

Resending the message as a receipt

Step 5: Bob signs the message with his private key, encrypts it with Alice's public key, and sends it back to Alice



Step 6: Alice decrypts the message with her private key, verifies with Bob's public key and recovers the message

6.1 $\{\{X\}_{k_{S,B}}\}_{k_{P,A}}, k_{S,A} = \{X\}_{k_{S,B}}$

6.2 $\{X\}_{k_{S,B}}, k_{P,B} = X$

Possible Attack

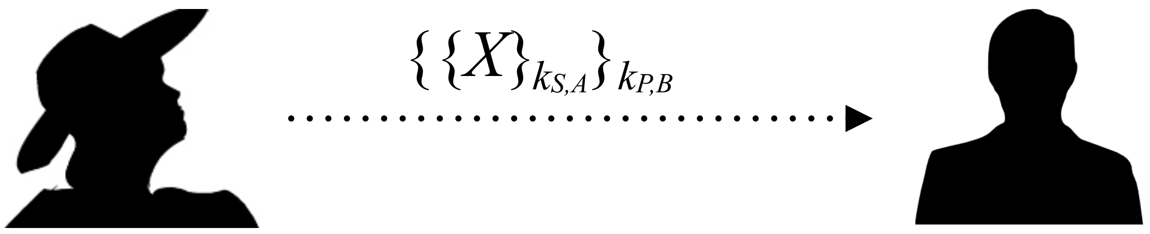
- Trouble can arise when the same algorithm is used for both encryption and digital signature verification
    - In these cases, the digital signature operation is the inverse of the encryption operation

Assume Mallory is a legitimate user with her own public and private key

Mallory records Alice's message to Bob in Step 2 of the protocol: $\{\{X\}_{k_{S,A}}\}_{k_{P,B}}$

Possible Attack

Mallory replays the message to Bob, claiming it came from her

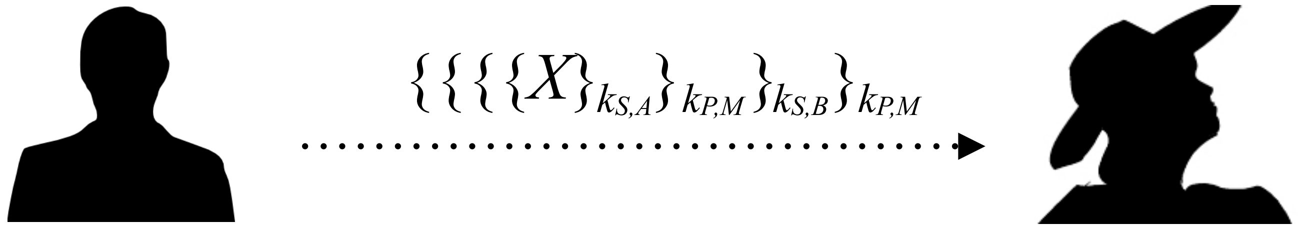


Bob decrypts the message, and tries to verify Mallory's signature by applying Mallory's public key. The resulting message is non-sense:

Result: $\{\{X\}_{k_{S,A}}\}_{k_{P,M}}$ $\longleftarrow$ Attack could be detected at this step

Possible Attack

Bob proceeds with the protocol and sends Mallory a receipt:



$$\{\{\{\{X\}_{k_{S,A}}\}_{k_{P,M}}\}_{k_{S,B}}\}_{k_{P,M}}$$

Mallory recovers X by:

1. Decrypting the message with her private key
2. Encrypting it with Bob's public key
3. Decrypting it again with her private key
4. Encrypting it with Alice's public key