Cryptography Lecture 10

Elliptic curve cryptography, key distribution and trust

Key length

Table 7.2: Key-size Equivalence.

Security (bits)	RSA	DLOG		EC
		field size	subfield	
48	480	480	96	96
56	640	640	112	112
64	816	816	128	128
80	1248	1248	160	160
112	2432	2432	224	224
128	3248	3248	256	256
160	5312	5312	320	320
192	7936	7936	384	384
256	15424	15424	512	512

Table 7.3: Effective Key-size of Commonly used RSA/DLOG Keys.

RSA/DLOG Key	Security (bits)
512	50
768	62
1024	73
1536	89
2048	103

From "ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012)"

Elliptic curves

- An elliptic curve is the set of solutions to the equation

$$y^2 = x^3 + ax^2 + bx + c$$

- These solutions are not ellipses, the name elliptic is used for historical reasons and has do to with the integrals used when calculating arc length in ellipses:

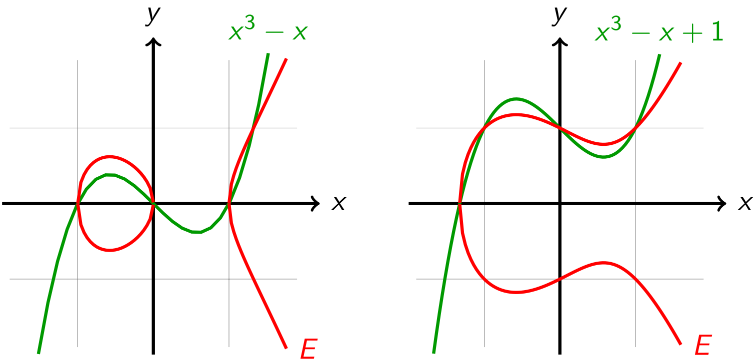
$$\int_a^b \frac{dx}{\sqrt{x^3 + ax^2 + bx + c}}$$

Elliptic curves

- An elliptic curve is the set

$$E = \{(x, y) : y^2 = x^3 + ax^2 + bx + c\}$$

- Examples:

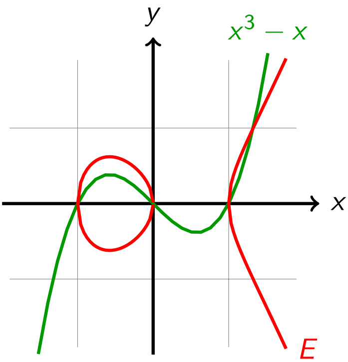


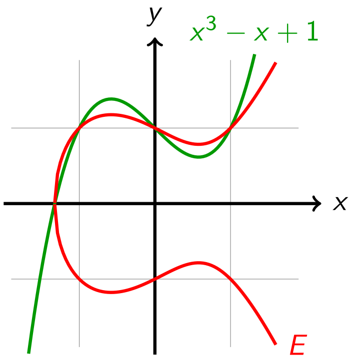
Elliptic curves

- Most of the time a "depressed" cubic is enough

$$E = \{(x, y) : y^2 = x^3 + bx + c\}$$

- Examples:



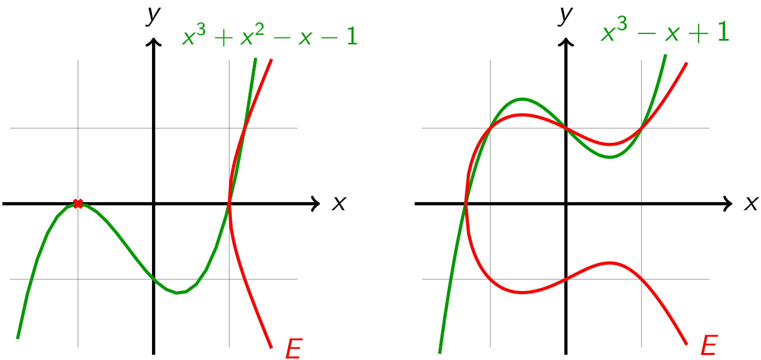


Elliptic curves

- You do not want "singular curves" with multiple roots

$$E = \{(x, y) : y^2 = x^3 + bx + c\}$$

- Examples:

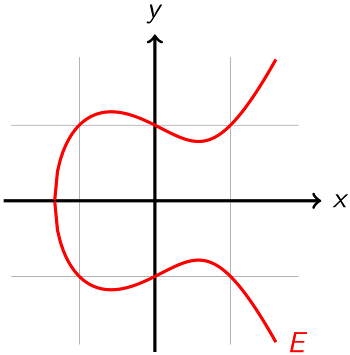


Elliptic curves

- An elliptic curve is the set

$$E = \{(x, y) : y^2 = x^3 + bx + c\}$$

- Previously we have used integers (mod p) and multiplication



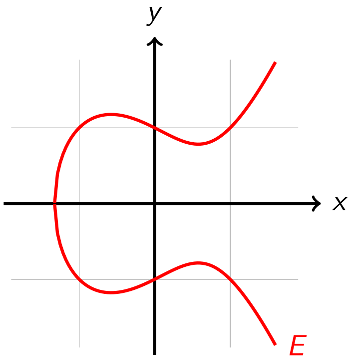
Elliptic curves

- An elliptic curve is the set

$$E = \{(x, y) : y^2 = x^3 + bx + c\}$$

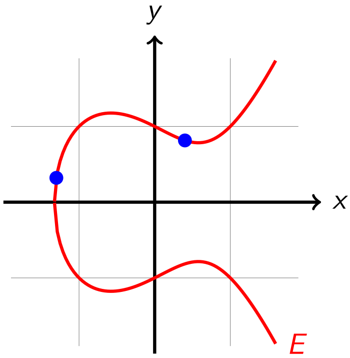
- Previously we have used the multiplicative group of integers mod p

- We need a group operation on points of E, we'll call it "addition"



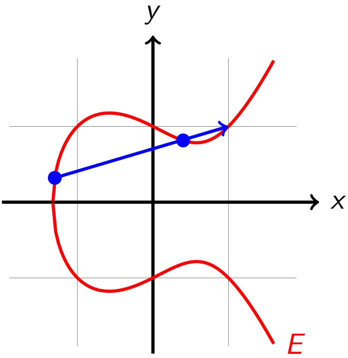
Addition on elliptic curves

- Given two elements in the group, construct a third



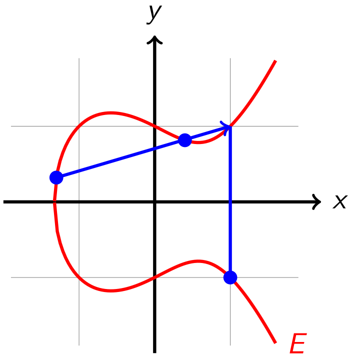
Addition on elliptic curves

- Given two elements in the group, construct a third
- Draw a straight line through the two points, it will intersect the elliptic curve in a third point.



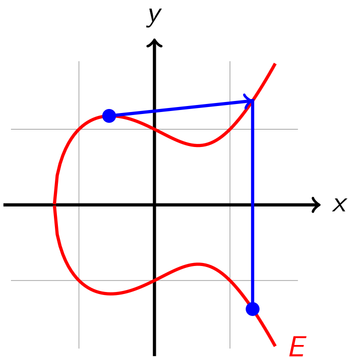
Addition on elliptic curves

- Given two elements in the group, construct a third
- Draw a straight line through the two points, it will intersect the elliptic curve in a third point. Mirror that in the x-axis



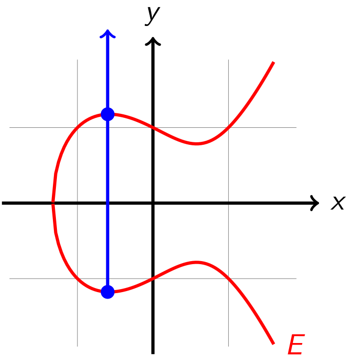
Addition on elliptic curves

- Given two elements in the group, construct a third
- Draw a straight line through the two points, it will intersect the elliptic curve in a third point. Mirror that in the x-axis
- If adding a point to itself, use the tangent line



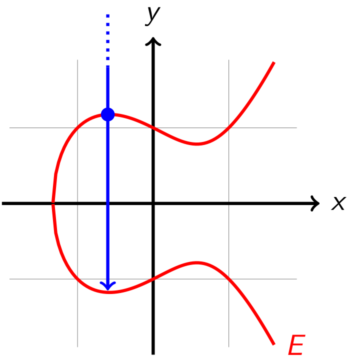
Addition on elliptic curves

- Given two elements in the group, construct a third
- There is one special case: if the line through the two points is vertical, it will not intersect the elliptic curve again
- We add the point (∞, ∞) to E
- This is the neutral element, the “0”



Addition on elliptic curves

- Given two elements in the group, construct a third
- The point (∞, ∞) to E is the neutral element, the "0"
- That is, $(\infty, \infty) + (x, y) = (x, y)$
- This also means that $-(x, y)$ is $(x, -y)$



Addition on elliptic curves

Addition law: On the elliptic curve

$$E = \{(x, y) : y^2 = x^3 + bx + c\},$$

$$(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$$

is calculated as follows:

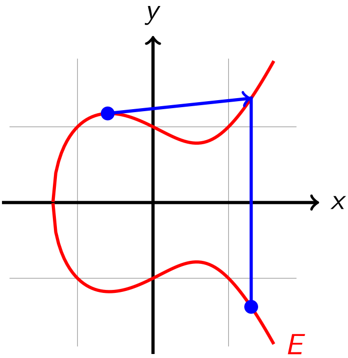
- If $(x_1, y_1) = (x_2, -y_2)$, then $(x_3, y_3) = (\infty, \infty)$
- If $(x_1, y_1) = (\infty, \infty)$, then $(x_3, y_3) = (x_2, y_2)$ (and the other way around)
- If $(x_1, y_1) = (x_2, y_2)$, then let $m = (3x_1^2 + b)/(2y_1)$, otherwise let $m = (y_2 - y_1)/(x_2 - x_1)$, and let

$$(x_3, y_3) = (m^2 - x_1 - x_2, m(x_1 - x_3) - y_1)$$

Multiplication on elliptic curves

- Multiplication with an integer is defined through repeated addition

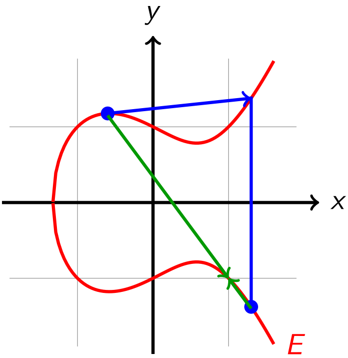
$$3(x, y) = (x, y) + (x, y) + (x, y)$$



Multiplication on elliptic curves

- Multiplication with an integer is defined through repeated addition

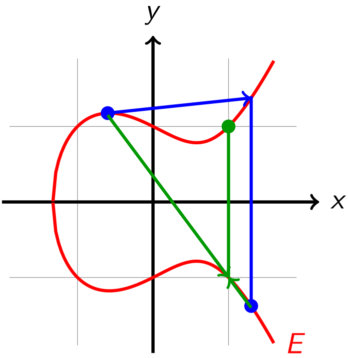
$$3(x, y) = (x, y) + (x, y) + (x, y)$$



Multiplication on elliptic curves

- Multiplication with an integer is defined through repeated addition

$$3(x, y) = (x, y) + (x, y) + (x, y)$$

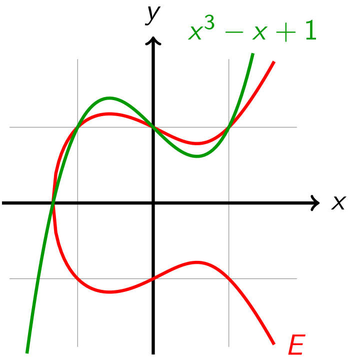


Discrete elliptic curves

- We want to have a discrete set of points. We arrange this by having coordinates mod p

$$E = \{(x, y) : y^2 = x^3 + bx + c \bmod p\}$$

- This is not so easy to draw in a diagram, remember, it is y^2 mod p



Discrete elliptic curves

- Example:

$$E = \{(x, y) : y^2 = x^3 + 4x + 4 \bmod 5\}$$

The points in E are

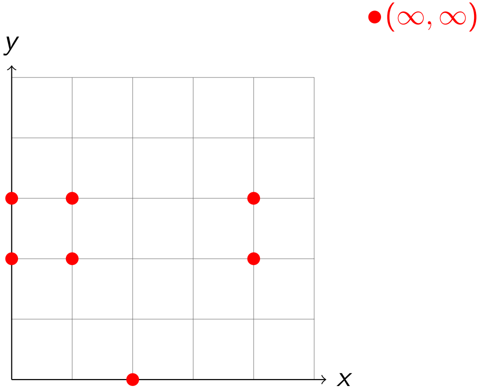
$$x = 0 \text{ gives } y^2 = 4 \text{ so that } y = 2 \text{ or } y = 3$$
$$x = 1 \text{ gives } y^2 = 9 = 4 \text{ so that } y = 2 \text{ or } y = 3$$
$$x = 2 \text{ gives } y^2 = 20 = 0 \text{ so that } y = 0$$
$$x = 3 \text{ gives } y^2 = 43 = 3, \text{ no square root}$$
$$x = 4 \text{ gives } y^2 = 84 = 4 \text{ so that } y = 2 \text{ or } y = 3$$
$$x = \infty \text{ gives } y = \infty$$

Discrete elliptic curves

- Example:

$$E = \{(x, y) : y^2 = x^3 + 4x + 4 \bmod 5\}$$

The points in E are



Elliptic curves

- Addition as we defined it still works on this set (but lines mod p need to be handled)
- We now have the group operations to use instead of integer multiplication and exponentiation
- Hasse's Theorem: The number of points N in an Elliptic curve E mod p obeys

$$p - 1 - 2\sqrt{p} < N < p - 1 + 2\sqrt{p}$$

Elliptic curves

- Addition as we defined it still works on this set (but lines mod p need to be handled)
- We now have the group operations to use instead of integer multiplication and exponentiation
- Hasse's Theorem: The number of points N in an Elliptic curve E mod p obeys

$$p - 1 - 2\sqrt{p} < N < p - 1 + 2\sqrt{p}$$

Addition on elliptic curves

Addition law: On the elliptic curve

$$E = \{(x, y) : y^2 = x^3 + bx + c\},$$

$$(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$$

is calculated as follows:

- If $(x_1, y_1) = (x_2, -y_2)$, then $(x_3, y_3) = (\infty, \infty)$
- If $(x_1, y_1) = (\infty, \infty)$, then $(x_3, y_3) = (x_2, y_2)$ (and the other way around)
- If $(x_1, y_1) = (x_2, y_2)$, then let $m = (3x_1^2 + b)/(2y_1)$, otherwise let $m = (y_2 - y_1)/(x_2 - x_1)$, and let

$$(x_3, y_3) = (m^2 - x_1 - x_2, m(x_1 - x_3) - y_1)$$

Addition on elliptic curves

Addition law: On the elliptic curve

$$E = \{(x, y) : y^2 = x^3 + bx + c\},$$

$$(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$$

is calculated as follows:

- If $(x_1, y_1) = (x_2, -y_2)$, then $(x_3, y_3) = (\infty, \infty)$
- If $(x_1, y_1) = (\infty, \infty)$, then $(x_3, y_3) = (x_2, y_2)$ (and the other way around)
- If $(x_1, y_1) = (x_2, y_2)$, then let $m = (3x_1^2 + b)/(2y_1)$, otherwise let $m = (y_2 - y_1)/(x_2 - x_1)$, and let

$$(x_3, y_3) = (m^2 - x_1 - x_2, m(x_1 - x_3) - y_1)$$

Elliptic curves

- Addition as we defined it still works on this set (but lines mod p need to be handled)
- We now have the group operations to use instead of integer multiplication and exponentiation
- Hasse's Theorem: The number of points N in an Elliptic curve E mod p obeys

$$p - 1 - 2\sqrt{p} < N < p - 1 + 2\sqrt{p}$$

Discrete Logarithms on elliptic curves

- Remember the discrete logarithm problem: given x and a primitive root g, find k so that

$$x = g^k \bmod p$$

- There is an analog on elliptic curves: given two points A and B on an elliptic curve, find k so that

$$B = kA = A + A + \ldots + A$$

- This might seem different, but is the equivalent problem. The only difference is the group operation *name* ("multiplication or "addition")

Discrete Logarithms on elliptic curves

- The discrete logarithm for elliptic curves: given two points A and B on an elliptic curve, find k so that

$$B = kA = A + A + \ldots + A$$

- There is an analog for the Polig-Hellman algorithm. This works well when the smallest integer n such that $nA = \infty$ has only small factors

Discrete Logarithms on elliptic curves

- The discrete logarithm for elliptic curves: given two points A and B on an elliptic curve, find k so that

$$B = kA = A + A + \ldots + A$$

- There is an analog for the Polig-Hellman algorithm
- The baby step-giant step algorithm works, but is impractical since it needs a lot of memory

Discrete Logarithms on elliptic curves

- The discrete logarithm for elliptic curves: given two points A and B on an elliptic curve, find k so that

$$B = kA = A + A + \ldots + A$$

- There is an analog for the Polig-Hellman algorithm
- The baby step-giant step algorithm is impractical
- But most importantly, there is no analog for the index calculus
  - Integer mod p index calculus is based on using small base numbers (not small exponents as in Polig-Hellman)
  - But there are no points on E that are closer to "0" than any other points, the distance to (∞, ∞) is the same for all other points

Key length

Table 7.2: Key-size Equivalence.

Security (bits)	RSA	DLOG		EC
		field size	subfield	
48	480	480	96	96
56	640	640	112	112
64	816	816	128	128
80	1248	1248	160	160
112	2432	2432	224	224
128	3248	3248	256	256
160	5312	5312	320	320
192	7936	7936	384	384
256	15424	15424	512	512

Table 7.3: Effective Key-size of Commonly used RSA/DLOG Keys.

RSA/DLOG Key	Security (bits)
512	50
768	62
1024	73
1536	89
2048	103

From "ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012)"

Trapdoor one-way functions

- A trapdoor one-way function is a function that is easy to compute but computationally hard to reverse
  - Easy to calculate xA from x
  - Hard to invert: to calculate x from xA
- A trapdoor one-way function has one more property, that with certain knowledge it *is* easy to invert, to calculate x from xA
- There is no proof that trapdoor one-way functions exist, or even real evidence that they can be constructed

Standard (m mod p) ElGamal encryption

- Choose a large prime p, and a primitive root α mod p. Also, take a random integer a and calculate $\beta = \alpha^a \bmod p$
- The public key is the values of p, α, and β, while the secret key is the value a
- Encryption uses a random integer k with $\gcd(k, p-1) = 1$, and the ciphertext is the pair $(\alpha^k, \beta^k m)$, both mod p
- Decryption is done with a, by calculating

$$(\alpha^k)^{-a}(\beta^k m) = (\alpha^{-ak})(\alpha^{ak} m) = m \bmod p$$

Elliptic curve ElGamal encryption

- Choose an elliptic curve E mod a large prime p, and a point α on E. Also, take a random integer a and calculate $\beta = a\alpha$
- The public key is E and the values of p, α, and β, while the secret key is the value a
- Encryption uses a random integer k, and the ciphertext is the pair $(k\alpha, k\beta + m)$
- Decryption is done with a, by calculating

$$-a(k\alpha) + (k\beta + m) = -ak\alpha + k(a\alpha) + m = m$$

Representing plaintext on elliptic curves

- Unfortunately, it is not simple to represent a given plaintext as a point on E
- Even worse, there is actually no polynomial time algorithm that can write down all points of an elliptic curve
- There is a method that will work with high probability:
  - The message m should be in the x-coordinate, but there is no guarantee that $m^3 + bm + c$ is a square mod p
  - Each number x has a probability of about $1/2$ that $x^3 + bx + c$ is a square, so put a few bits at the end of m and run through all possible values
  - If the number of possible values is K, the risk of failure is 2^{-K}

Standard (integer mod p) Diffie-Hellman key exchange

- Use two one-way functions f and g: exponentiation mod p (of a primitive root α), the symmetry is

$$(\alpha^a)^b = (\alpha^b)^a \bmod p$$

- This cannot be used for encryption/signing because one does not recover a or b.
- But it can be used for key exchange:
parameters p and α
  - Alice takes a secret random a and makes α^a public
  - Bob takes a secret random b and makes α^b public
  - Both can now create $k = (\alpha^a)^b = (\alpha^b)^a \bmod p$

Elliptic curve Diffie-Hellman key exchange

- Use two one-way functions f and g: multiplication on an elliptic curve E (of a point α), the symmetry is

$$b(a\alpha) = a(b\alpha)$$

- This cannot be used for encryption/signing because one does not recover a or b.
- But it can be used for key exchange: parameters E, p and α
  - Alice takes a secret random a and makes $a\alpha$ public
  - Bob takes a secret random b and makes $b\alpha$ public
  - Both can now create $k = b(a\alpha) = a(b\alpha)$

Standard (mod p) ElGamal signatures

- Choose a large prime p, and a primitive root α mod p. Also, take a random integer a and calculate $\beta = \alpha^a \bmod p$
- The public key is the values of p, α, and β, while the secret key is the value a
- Signing uses a random integer k with $\gcd(k, p-1) = 1$, and the signature is the pair (r, s) where

$$r = \alpha^k \bmod p$$
$$s = k^{-1}(m - ar) \bmod (p-1)$$

- Verification is done comparing $\beta^r r^s$ and α^m mod p, since

$$\beta^r r^s = \alpha^{ar}\alpha^{k(m-ar)/k} = \alpha^m \bmod p$$

Elliptic curve ElGamal signatures

- Choose an elliptic curve E mod a large prime p, and a point α on E. Also, take a random integer a and calculate $\beta = a\alpha$
- The public key is E and the values of p, α, and β, while the secret key is the value a
- Signing uses a random integer k with $\gcd(k, n) = 1$ where n is the number of points on E. The signature is created by inverting k mod n and forming the pair (r, s) as

$$
\begin{aligned}
r &= k\alpha \\
s &= k^{-1}(m - ar_x)
\end{aligned}
$$

- Verification is done comparing $r_x\beta + sr$ and $m\alpha$, since

$$
\begin{aligned}
r_x\beta + sr &= r_x(a\alpha) + \big(k^{-1}(m - ar_x)\big)(k\alpha) \\
&= r_x(a\alpha) + m\alpha - ar_x\alpha = m\alpha
\end{aligned}
$$

Trapdoor one-way functions

A trapdoor one-way function is a function that is easy to compute but computationally hard to reverse

- Easy to calculate $f(x)$ from x
- Hard to invert: to calculate x from $f(x)$

A trapdoor one-way function has one more property, that with certain knowledge it *is* easy to invert, to calculate x from $f(x)$

There is no proof that trapdoor one-way functions exist, or even real evidence that they can be constructed. Examples:

- RSA (factoring)
- Knapsack (NP-complete but insecure with trapdoor)
- Diffie-Hellman + ElGamal (discrete log)
- EC Diffie-Hellman + EC ElGamal (EC discrete log)

Key Management

- The first key in a new connection or association is *always* delivered via a courier
- Once you have a key, you can use that to send new keys
- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can exchange a key via Trent (provided they both trust Trent)

Key distribution center

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can exchange a key via Trent (provided they both trust Trent)

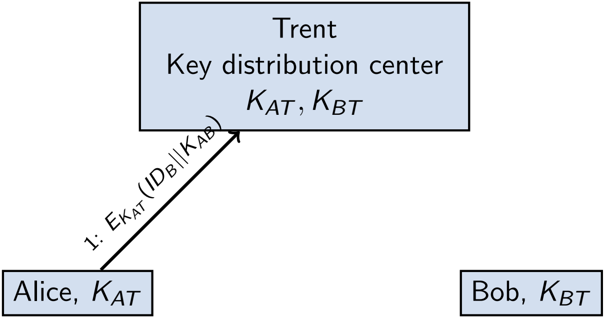
Trent
Key distribution center
K_{AT}, K_{BT}

Alice, K_{AT}

Bob, K_{BT}

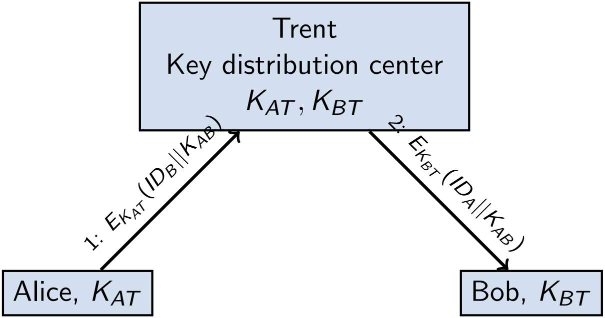
Key distribution center

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can exchange a key via Trent (provided they both trust Trent)



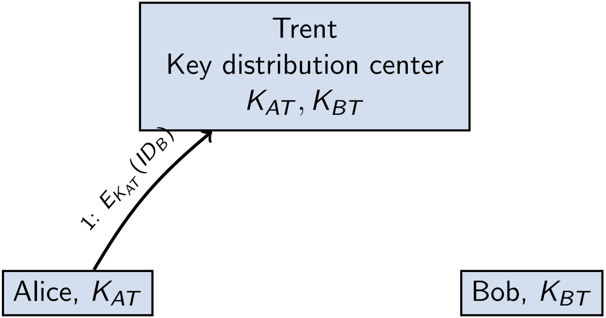
Key distribution center

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can exchange a key via Trent (provided they both trust Trent)



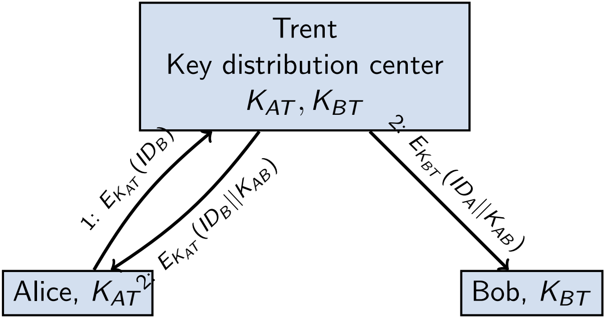
Key distribution center, key server

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can receive a key from Trent (provided they both trust Trent)



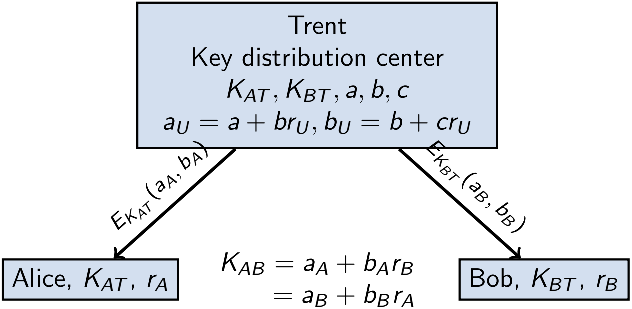
Key distribution center, key server

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can receive a key from Trent (provided they both trust Trent)



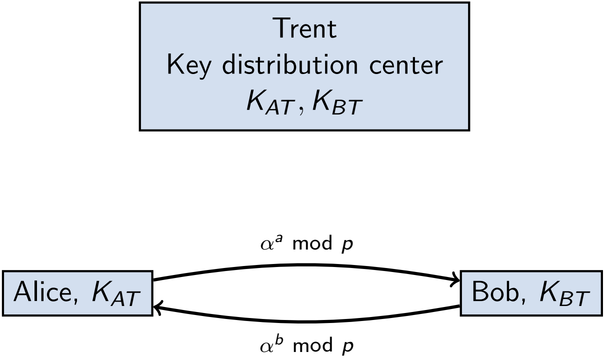
Key distribution center, Blom key pre-distribution

- If Alice shares a key with Trent and Trent shares a key with Bob, and Alice and Bob each have a public id r_A, r_B, they can recieve key-generation info from Trent (provided they both trust Trent)



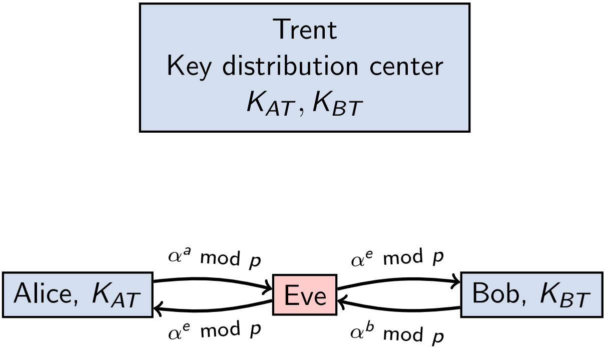
Key distribution center, Station-To-Station (STS) protocol

- What about Diffie-Hellman key exchange?



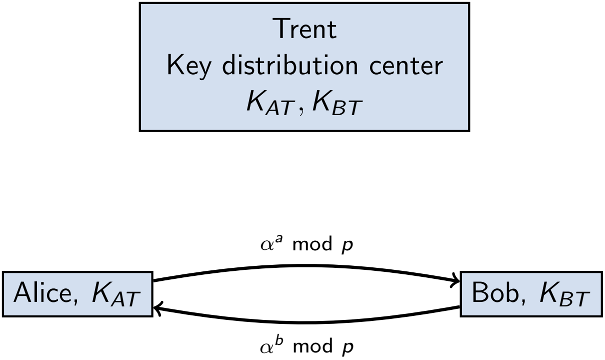
Key distribution center, Station-To-Station (STS) protocol

- What about Diffie-Hellman key exchange?
- Eve can do an "intruder-in-the-middle"



Key distribution center, Station-To-Station (STS) protocol

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can use Trent to verify that they exchange key with the right person



Key distribution center, Station-To-Station (STS) protocol

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can use Trent to verify that they exchange key with the right person

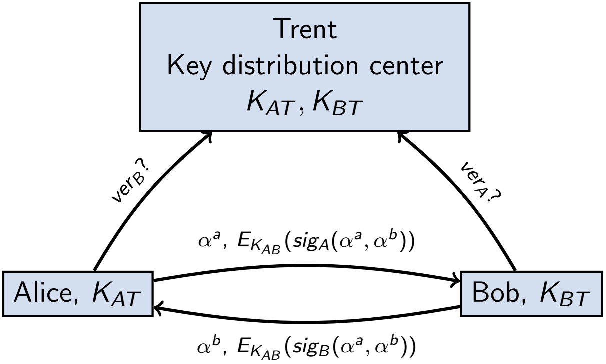
Trent
Key distribution center
K_{AT}, K_{BT}

$$\text{Alice}, K_{AT} \xrightarrow{\alpha^a,\ E_{K_{AB}}(sig_A(\alpha^a, \alpha^b))} \text{Bob}, K_{BT}$$

$$\text{Alice}, K_{AT} \xleftarrow[\alpha^b,\ E_{K_{AB}}(sig_B(\alpha^a, \alpha^b))]{} \text{Bob}, K_{BT}$$

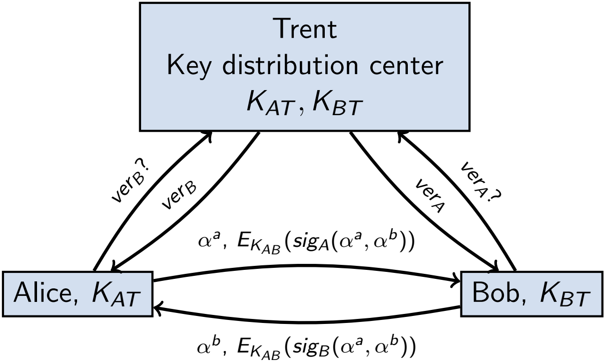
Key distribution center, Station-To-Station (STS) protocol

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can use Trent to verify that they exchange key with the right person



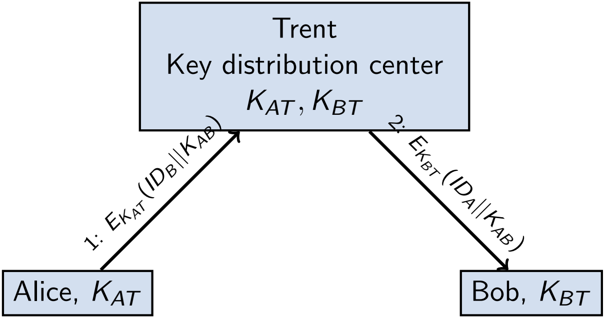
Key distribution center, Station-To-Station (STS) protocol

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can use Trent to verify that they exchange key with the right person



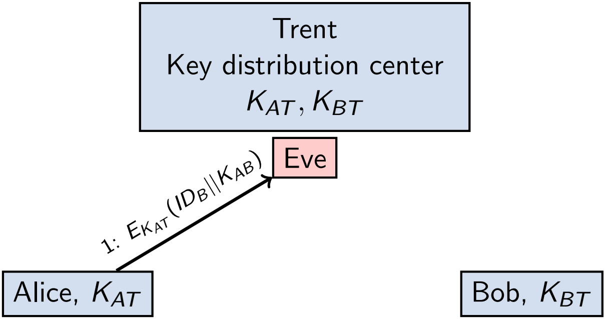
Key distribution center

- If Alice shares a key with Trent and Trent shares a key with Bob, then Alice and Bob can exchange a key via Trent (provided they both trust Trent)



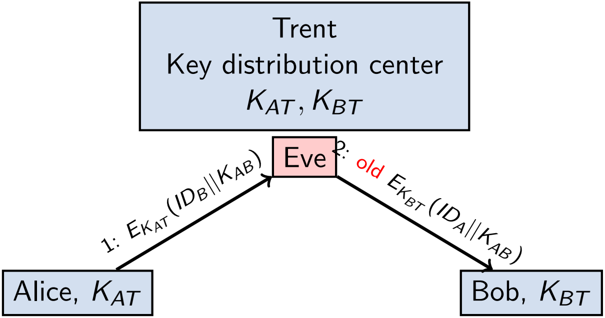
Key distribution center, replay attacks

- But perhaps Eve has broken a previously used key, and intercepts Alice's request



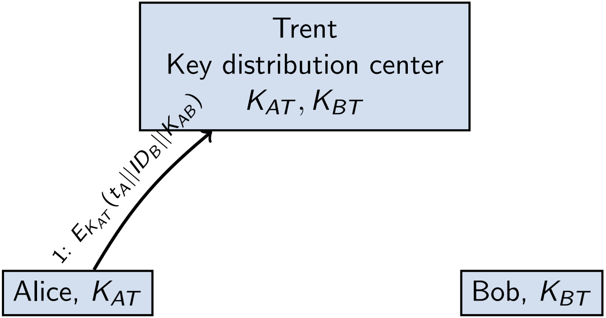
Key distribution center, replay attacks

- But perhaps Eve has broken a previously used key, and intercepts Alice's request
- Then she can fool Bob into communicating with her



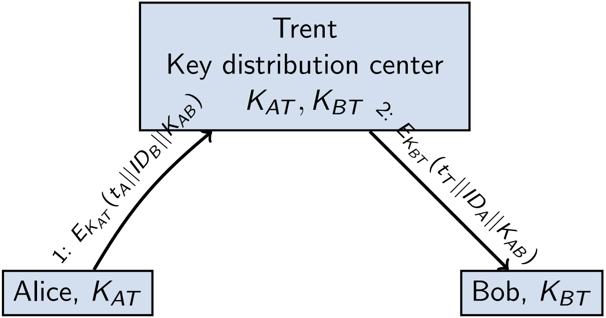
Key distribution center, wide-mouthed frog

- Alice and Trent add time stamps to prohibit the attack



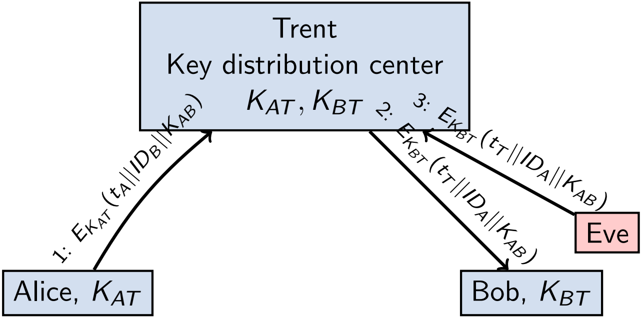
Key distribution center, wide-mouthed frog

- Alice and Trent add time stamps to prohibit the attack



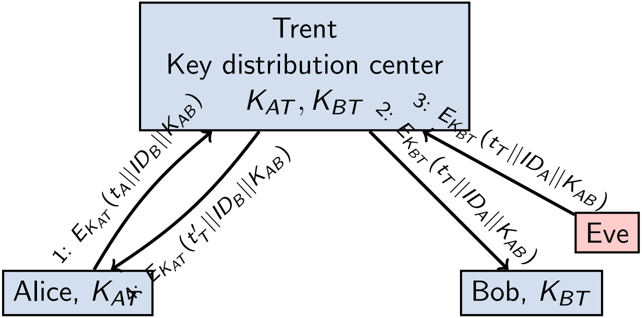
Key distribution center, wide-mouthed frog

- Alice and Trent add time stamps to prohibit the attack
- But now, Eve can pretend to be Bob and make a request to Trent



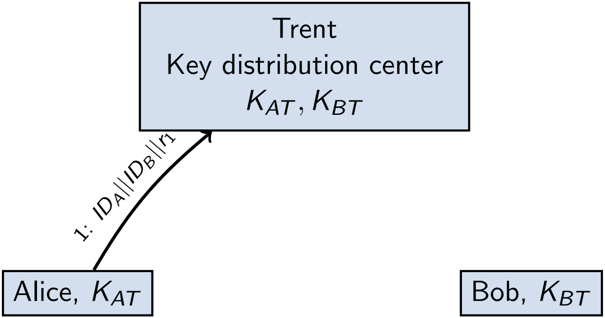
Key distribution center, wide-mouthed frog

- Alice and Trent add time stamps to prohibit the attack
- But now, Eve can pretend to be Bob and make a request to Trent, who will forward the key to Alice



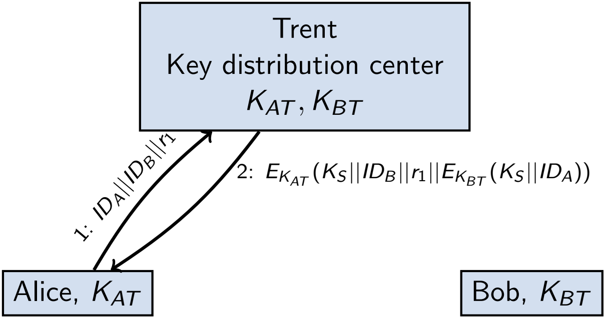
Key distribution center, Needham-Schroeder key agreement

- Another variation is to use nonces to prohibit the replay attack



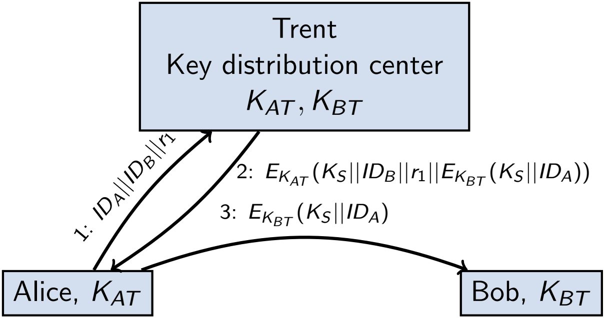
Key distribution center, Needham-Schroeder key agreement

- Another variation is to use nonces to prohibit the replay attack



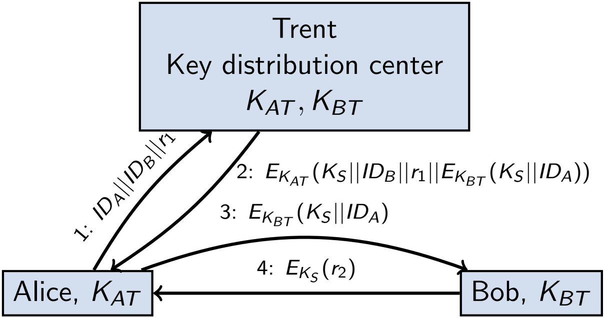
Key distribution center, Needham-Schroeder key agreement

- Another variation is to use nonces to prohibit the replay attack



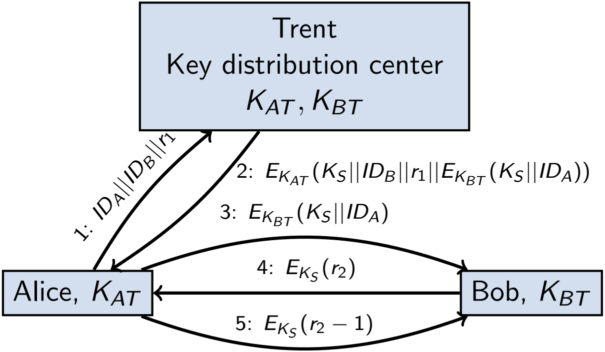
Key distribution center, Needham-Schroeder key agreement

- Another variation is to use nonces to prohibit the replay attack



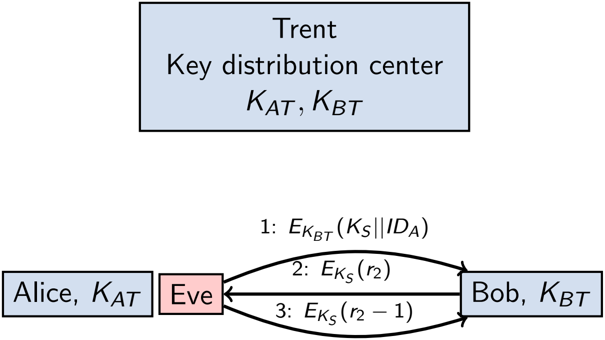
Key distribution center, Needham-Schroeder key agreement

- Another variation is to use nonces to prohibit the replay attack

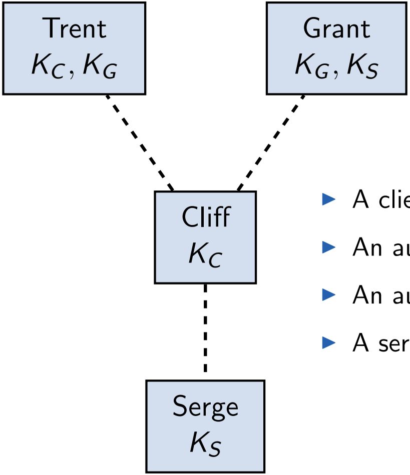


Key distribution center, Needham-Schroeder key agreement

- Another variation is to use nonces to prohibit the replay attack
- If Eve ever breaks one session key, she can get Bob to reuse it

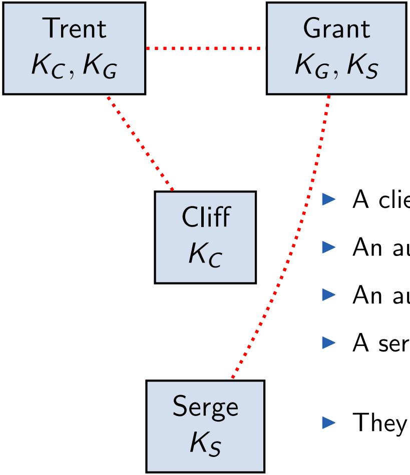


Kerberos



- A client, Cliff
- An authentication server, Trent
- An authorization server, Grant
- A service server, Serge

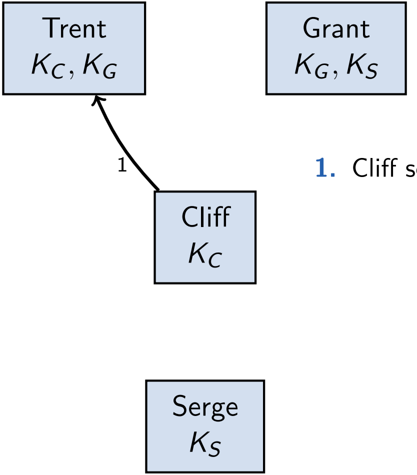
Kerberos



- A client, Cliff
- An authentication server, Trent
- An authorization server, Grant
- A service server, Serge

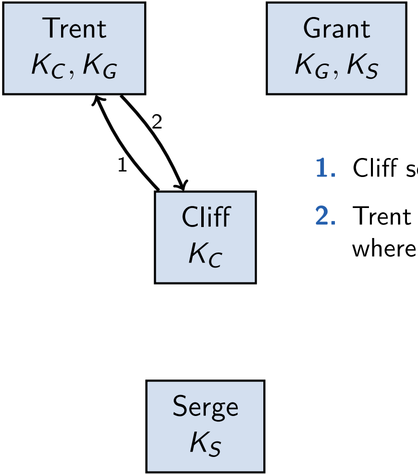
- They share keys K_C, K_G, K_S

Kerberos



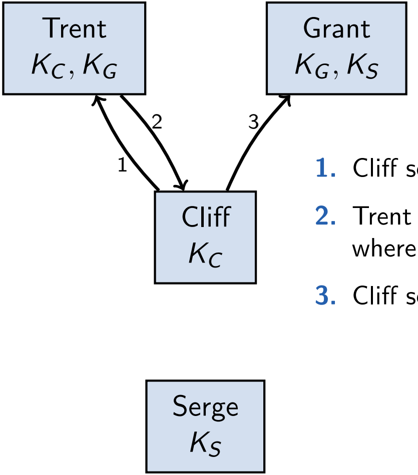
1. Cliff sends Trent $ID_C || ID_G$

Kerberos



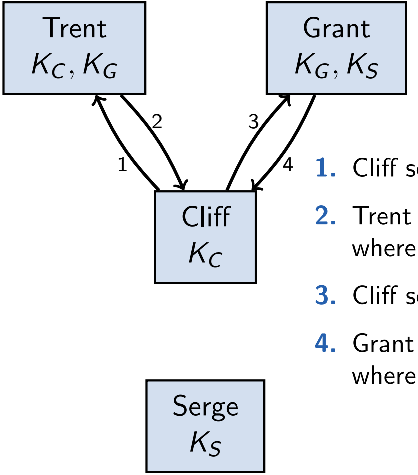
1. Cliff sends Trent $ID_C || ID_G$
2. Trent responds width $E_{K_C}(K_{CG}) || TGT$ where $TGT = ID_G || E_{K_G}(ID_C || t_1 || K_{GC})$

Kerberos



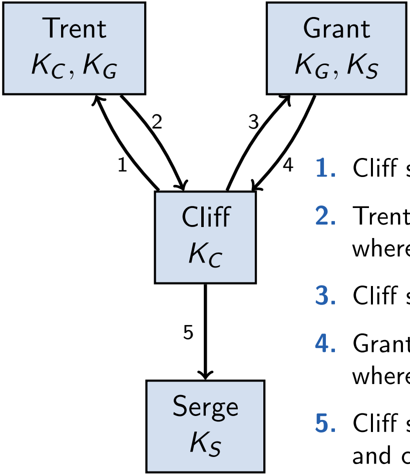
1. Cliff sends Trent $ID_C || ID_G$
2. Trent responds width $E_{K_C}(K_{CG}) || TGT$ where $TGT = ID_G || E_{K_G}(ID_C || t_1 || K_{GC})$
3. Cliff sends Grant $E_{K_{CG}}(ID_C || t_2) || TGT$

Kerberos



1. Cliff sends Trent $ID_C || ID_G$
2. Trent responds width $E_{K_C}(K_{CG}) || TGT$ where $TGT = ID_G || E_{K_G}(ID_C || t_1 || K_{GC})$
3. Cliff sends Grant $E_{K_{CG}}(ID_C || t_2) || TGT$
4. Grant responds with $E_{K_{CG}}(K_{CS}) || ST$ where $ST = E_{K_S}(ID_C || t_3 || t_{\text{expir.}} || K_{CS})$

Kerberos



1. Cliff sends Trent $ID_C || ID_G$
2. Trent responds width $E_{K_C}(K_{CG}) || TGT$ where $TGT = ID_G || E_{K_G}(ID_C || t_1 || K_{GC})$
3. Cliff sends Grant $E_{K_{CG}}(ID_C || t_2) || TGT$
4. Grant responds with $E_{K_{CG}}(K_{CS}) || ST$ where $ST = E_{K_S}(ID_C || t_3 || t_{\text{expir.}} || K_{CS})$
5. Cliff sends Serge $E_{K_{CS}}(ID_C || t_4) || ST$ and can then use Serge's services

Public key distribution

- Public key distribution uses a Public Key Infrastructure (PKI)

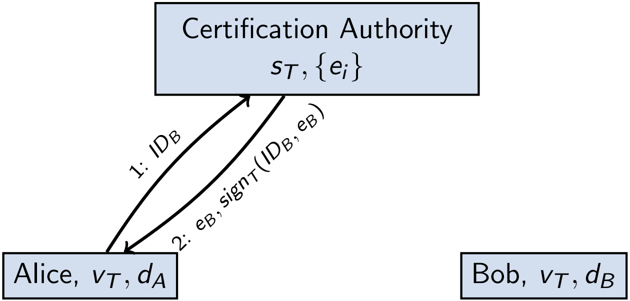
Certification Authority
$s_T, \{e_i\}$

Alice, v_T, d_A

Bob, v_T, d_B

Public key distribution, using Certification Authorities

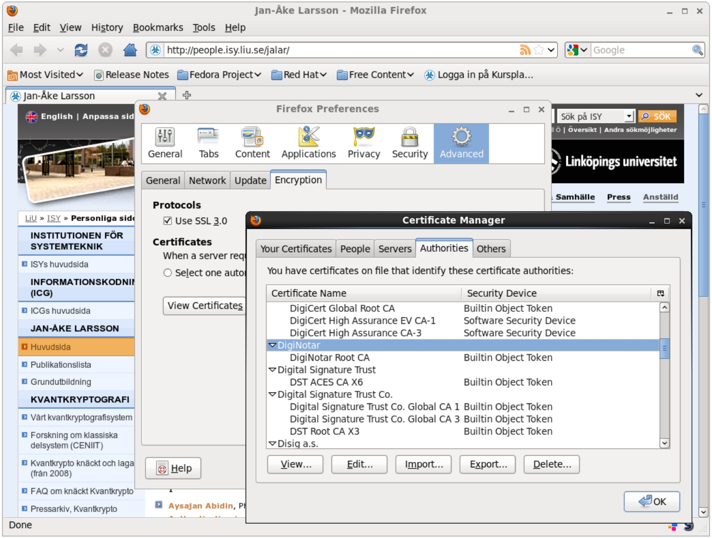
- Public key distribution uses a Public Key Infrastructure (PKI)
- Alice sends a request to a Certification Authority (CA) who responds with a certificate, ensuring that Alice uses the correct key to communicate with Bob



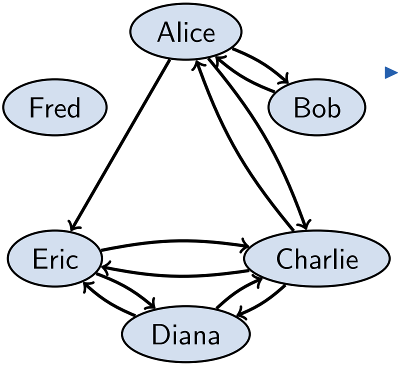
Public key distribution, using X.509 certificates

- The CAs often are commercial companies, that are assumed to be trustworthy
- Many arrange to have the root certificate packaged with IE, Mozilla, Opera,...
- They issue certificates for a fee
- They often use Registration Authorities (RA) as sub-CA for efficiency reasons

Public key distribution, X.509 certificates in your browser

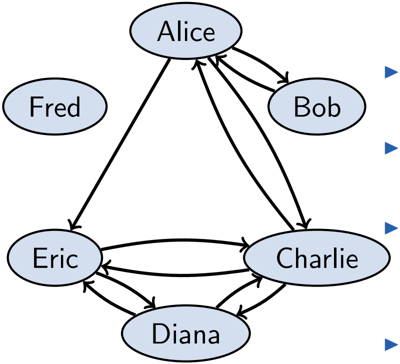


Public key distribution, using web of trust



- No central CA
- Users sign each other's public key (hashes)
- This creates a "web of trust"

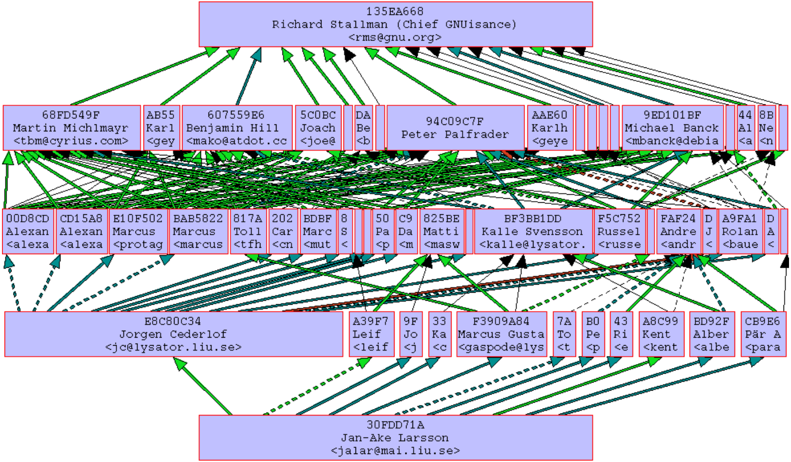
Public key distribution, using web of trust (PGP and GPG)



- No central CA
- Users sign each other's public key (hashes)
- This creates a "web of trust"

- Each user keeps a keyring with the keys (s)he has signed
- The secret key is stored on a secret keyring, on h{er,is} computer
- The public key(s) and their signatures are uploaded to key servers

Public key distribution, a web-of-trust path



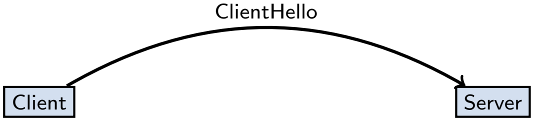
http://www.lysator.liu.se/~jc/wotsap/

Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

- This is a client-server handshake procedure to establish key
- The server (but not the client) is authenticated (by its certificate)

Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

ClientHello: highest TLS protocol version, random number, suggested public key systems + symmetric key systems + hash functions + compression algorithms



Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

ClientHello: highest TLS protocol version, random number, suggested public key systems + symmetric key systems + hash functions + compression algorithms

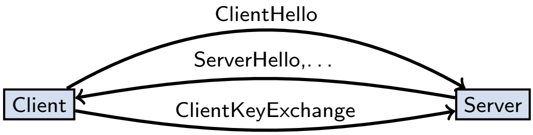
ServerHello, Certificate, ServerHelloDone: chosen protocol version, a (different) random number, system choices, public key

Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

ClientHello: highest TLS protocol version, random number, suggested public key systems + symmetric key systems + hash functions + compression algorithms

ServerHello, Certificate, ServerHelloDone: chosen protocol version, a (different) random number, system choices, public key

ClientKeyExchange: PreMasterSecret, encrypted with the server's public key



Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

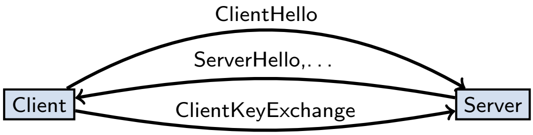
ClientHello: highest TLS protocol version, random number, suggested public key systems + symmetric key systems + hash functions + compression algorithms

ServerHello, Certificate, ServerHelloDone: chosen protocol version, a (different) random number, system choices, public key

ClientKeyExchange: PreMasterSecret, encrypted with the server's public key

(Master secret): creation of master secret using a pseudorandom function, with the PreMasterSecret as seed

(Session keys): session keys are created using the master secret, different keys for the two directions of communication



Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

ClientHello: highest TLS protocol version, random number, suggested public key systems + symmetric key systems + hash functions + compression algorithms

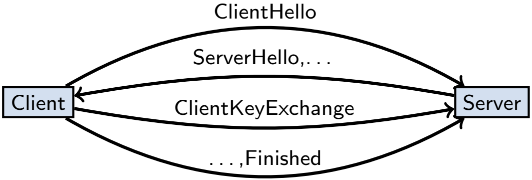
ServerHello, Certificate, ServerHelloDone: chosen protocol version, a (different) random number, system choices, public key

ClientKeyExchange: PreMasterSecret, encrypted with the server's public key

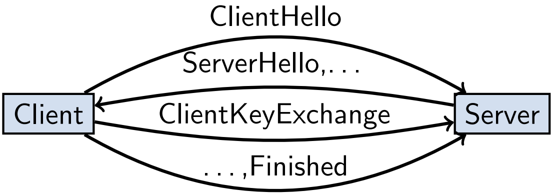
(Master secret): creation of master secret using a pseudorandom function, with the PreMasterSecret as seed

(Session keys): session keys are created using the master secret, different keys for the two directions of communication

ChangeCipherSpec, Finished authenticated and encrypted, containing a MAC for the previous handshake messages

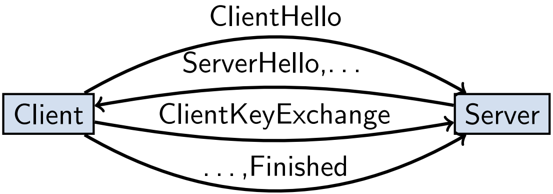


Secure Sockets Layer (SSL) and Transport Layer Security (TLS)



- SSL 1.0 (no public release), 2.0 (1995), 3.0 (1996), originally by Netscape
- TLS 1.0 (1999), changes that improve security, among other things how random numbers are chosen
  - Sensitive to CBC vulnerability discovered 2002, demonstrated by BEAST attack 2011
  - Current problem: TLS 1.0 is fallback if either end does not support higher versions

Secure Sockets Layer (SSL) and Transport Layer Security (TLS)



- TLS 1.1 (2006), added protection against CBC attacks by explicit IV specification
- TLS 1.2 (2008), e.g., change MD5-SHA1 to SHA256
- Later (2011), never fall back to SSL 2.0