Public Key Cryptographic Primitives

István Zsolt BERTA
istvan@berta.hu

PKI lectures

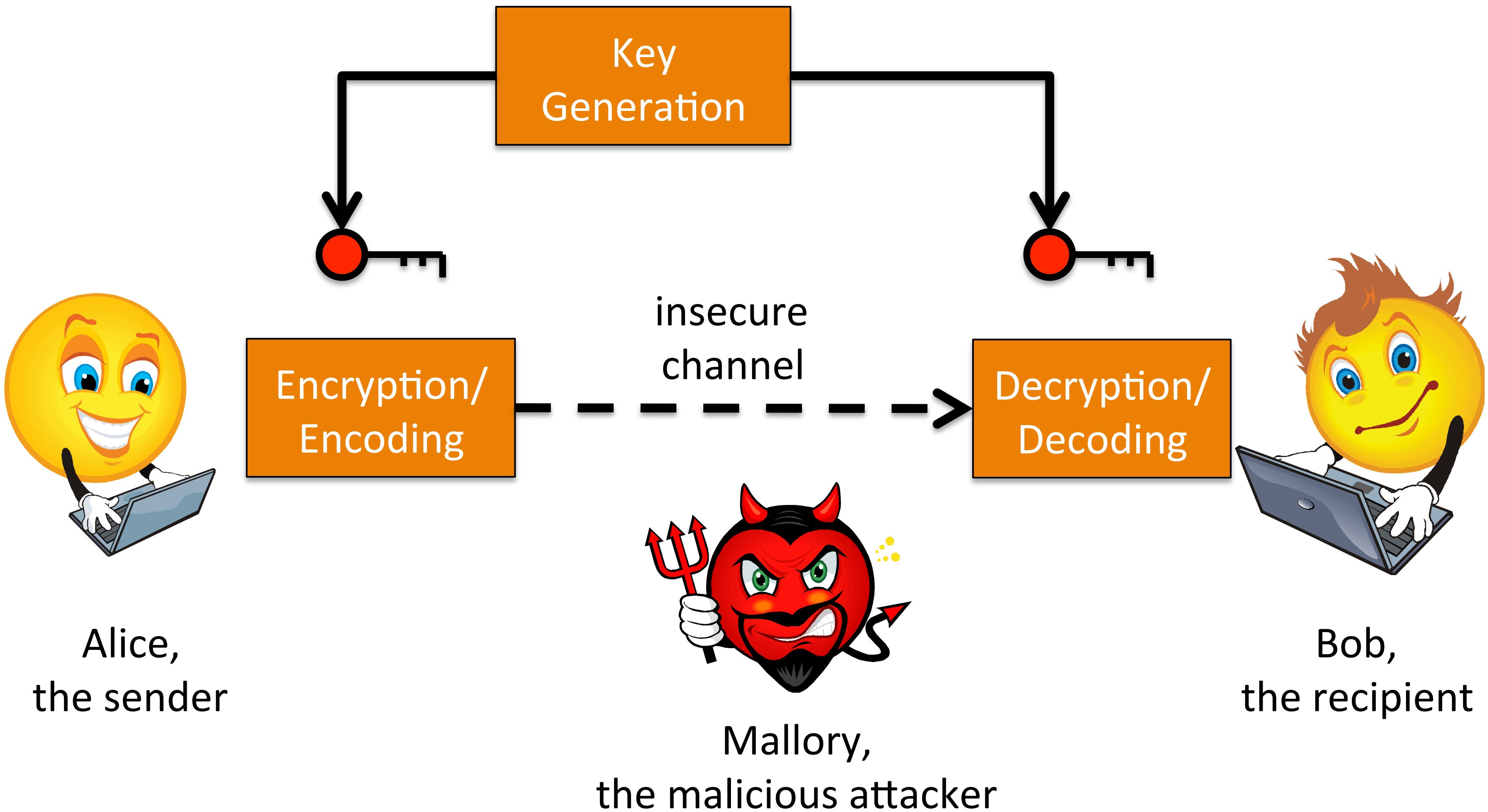
1. **Public key cryptography primitives**
2. Certificates, Certificate Authorities, Certification Paths
3. Electronic signatures: signature creation & validation
4. Information security management at CAs
5. PKI business

Public Key Crypto Primitives - Contents

- Public Key Cryptography
- RSA algorithm
- ECC algorithm

Public Key Cryptography

Symmetric Key Cryptography

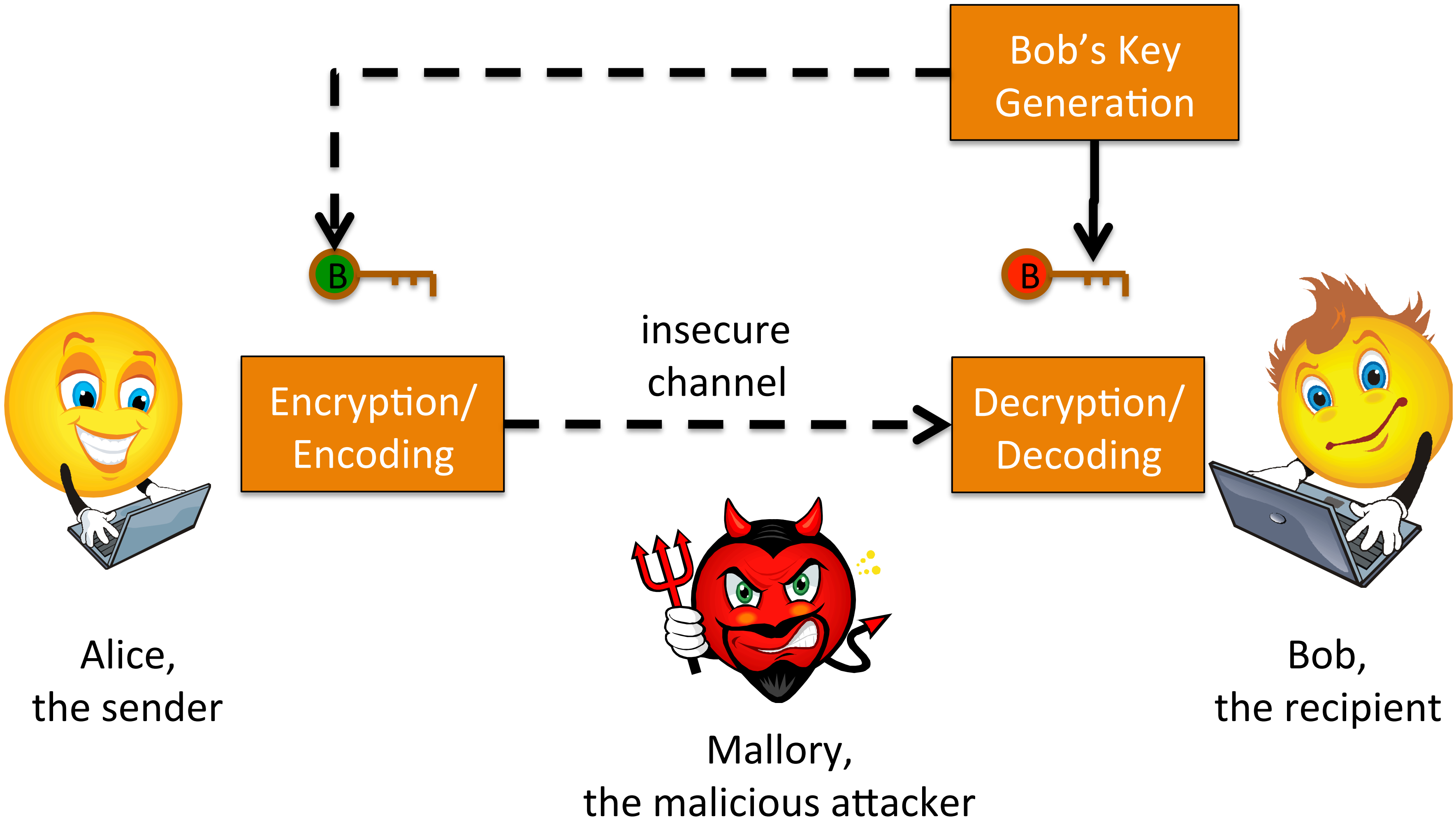


Symmetric Key Cryptography

- Same key is used for encryption and decryption
- Symmetric key algorithms are fast and
  short keys (e.g. 256 bits) can provide good security
- A symmetric key must be kept confidential; if the attacker learns the key, he may decrypt messages or sign messages on behalf of the sender
- Symmetric keys must be transmitted via a secure channel, and need to be a **shared secret** of the sender and recipient
- Example algorithms: AES, 3DES, RC4, Twofish, …

Public Key Cryptography

a.k.a. Asymmetric Key Cryptography



Public Key Cryptography

a.k.a. Asymmetric Key Cryptography

- Encryption and decryption are performed with different keys
- In fact, the key has two parts:
  - one part can be used for encryption/verification only, this can even be **public**
  - the other part can be used for decryption/signature, this must be kept **private**
- Only the public key needs to be transmitted to the recipient, and this does not need a secure channel
- There is no need to have shared secret between sender and recipient → this makes key management easier
- Public key cryptography is slower than symmetric key crypto and require longer (e.g. 2048 bits) keys for similar security

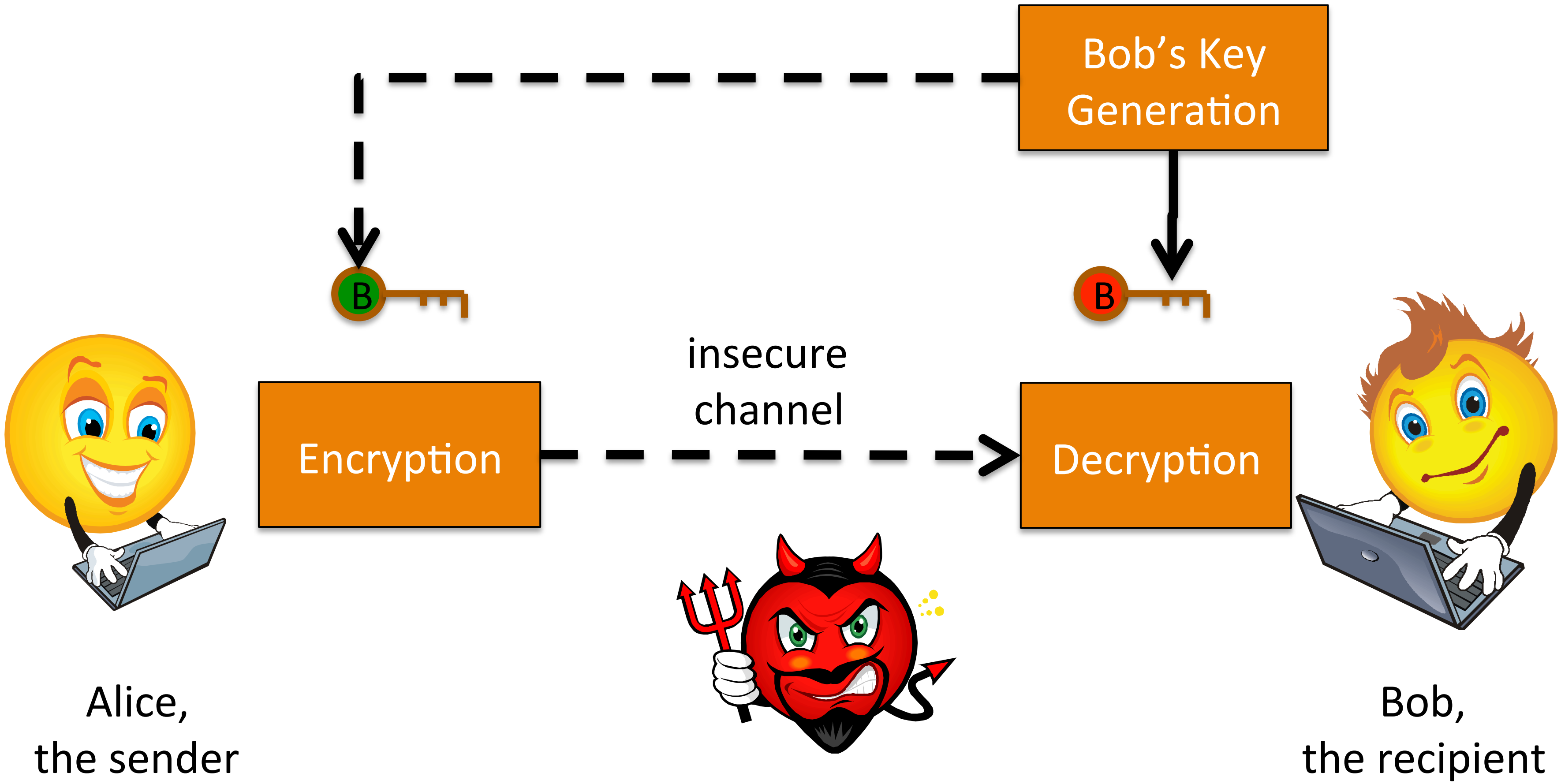
Public key and private key

- The public and the private key must be interlinked, so that
  - messages encrypted with the public key can be decrypted with the corresponding private key; and
  - messages signed with the private key can be verified with the corresponding public key
- The must not be an efficient method for computing the private key from the public key
- Most public key algorithms are based on mathematical problems with the above properties, e.g.:
  - RSA: Integer Factorization Problem (IFP)
  - ECC: Elliptic Curve Discrete Logarithm Problem (ECDLP)

Digital Signature

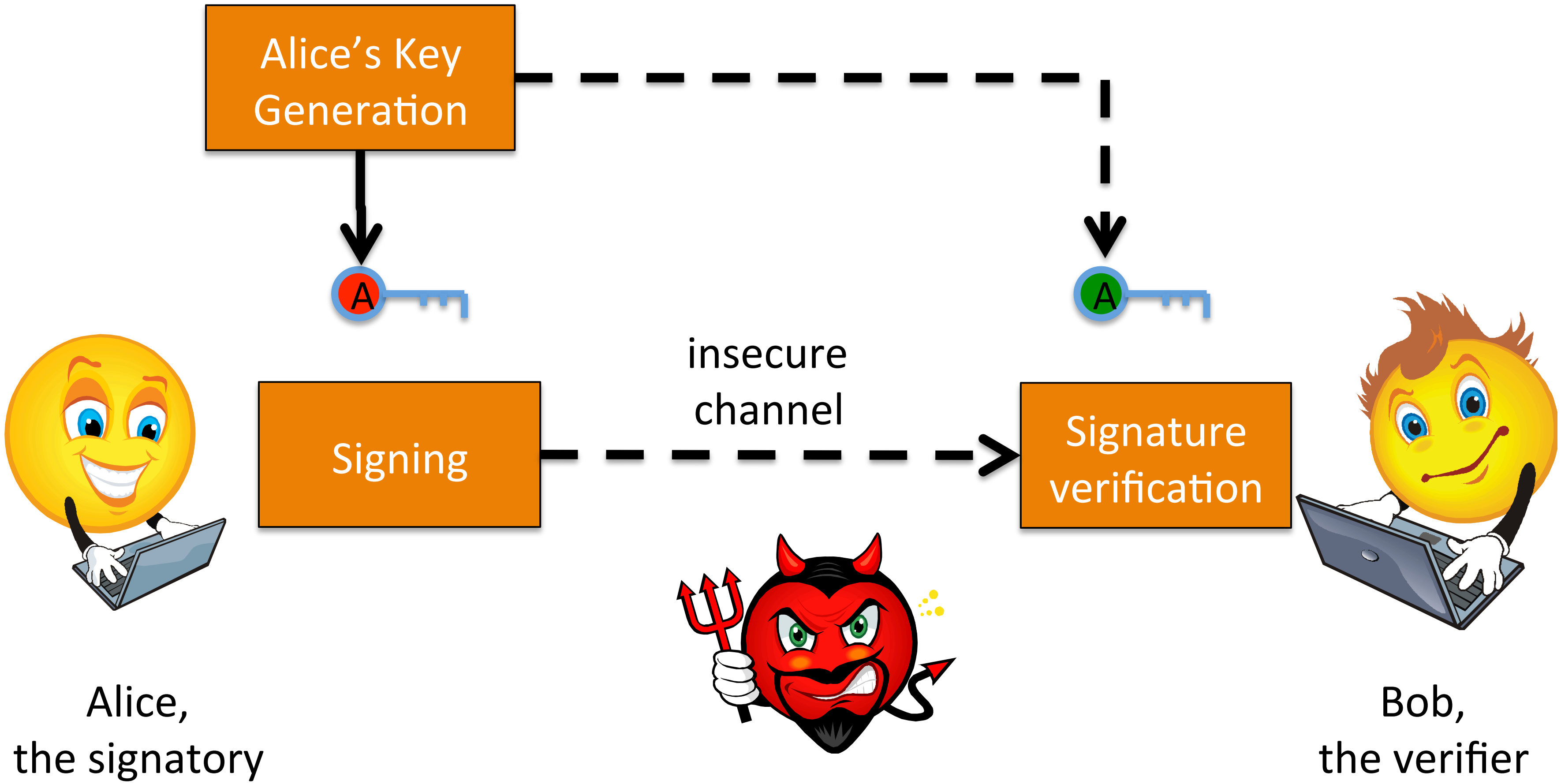
- Public key cryptosystems allow the concept of digital signature
- A message encoded with Alice's private key is *signed*:
  - Such an encoded message cannot be computed without Alice's private key, and
  - anyone can verify this with Alice's public key
  - The signature proves that Alice signed the given message and that it had not altered since she signed it
- The signature is usually transmitted together with the cleartext message
- Note that the signature does not provide confidentiality

Sending an encrypted message



Encryption is performed with the recipient's public key; the recipient can decrypt the message with their private key

Sending a digitally signed message



The sender/signatory signs the message with their private key; anyone (any recipient) can verify that the message not altered after it had been signed by the signatory

Summary: Symmetric vs Asymmetric

Symmetric key solutions:

- fast
- small keys (e.g. 256 bits)
- distribution of keys is a challenge as a secure channel is needed

Asymmetric key (public key) solutions:

- slower
- long keys (e.g. 2048 bits)
- distribution of public keys does not need a secure channel
- signature is possible

Typical combinations

1. Use public key crypto for exchanging symmetric keys; then use these symmetric keys for bulk encryption – e.g. TLS, IPSEC
2. /Encrypt the long message with a random symmetric key; encrypt the symmetric key only with the public key(s) of the recipient(s) – e.g. SMIME/
3. Compute a hash of the message and sign the hash only with the private key – most digital signature solutions work this way

RSA algorithm

Factorization

65536	2
32768	2
16384	2
8192	2
4096	2
2048	2
1024	2
512	2
256	2
. . .	…
4	2
2	2
1	

65537	…, …, 65537
1	

- This is a prime!

Integer factorization is a HARD problem

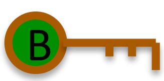
- No algorithm is known that can efficiently factorize an any large composite number
- IFP: Integer Factorization Problem
- RSA is a cryptosystem based on the IFP, it implements both encryption and signature
- Ron Rivest, Adi Shamir and Leonard Adleman - 1977

RSA (Rivest-Shamir-Aldeman) alg. in a nutshell

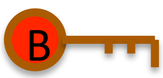
1. Choose two random prime numbers: p and q
2. Compute their product: $\mathbf{m} = p * q$
3. Compute $\Phi(m) = (p-1) * (q-1)$
4. Select number $\mathbf{e}$ to be relative prime to $\Phi(m)$.
5. Compute number $\mathbf{d}$, so that $\mathbf{e} * \mathbf{d} = 1 \pmod{\Phi(m)}$

For any number x: $(x^{\mathbf{e}})^{\mathbf{d}} = x \pmod{m}$

Bob's public key: $\mathbf{m}$ and $\mathbf{e}$



Bob's private key: $\mathbf{d}$



RSA key generation

- RSA key size is the size of the modulus (m)
- Select two random large (m/2 bits) random numbers 1xxxxxxxxxx….xxxxxx1
- Check if they are prime, repeat until two primes are found
  - in practice, randomized primality testing algorithms (e.g. Miller-Rabin) are used, chance of a composite number passing the test can be made arbitrarily low
- Public exponent e is usually a fixed number
  - a low e allows quick operations with a public key
  - primes with a low number of 1s in their binary representation
  - previously: 3, now: 65537
- Private exponent d can be computed using the extended Euclidean algorithm

Toy RSA (with small numbers)

1. Choose two random prime numbers: $p = 5$ and $q = 11$
2. Compute their product: $m = p * q = 5*11 = 55$
3. Compute $\Phi(m) = (p-1) * (q-1) = 4*10 = 40$
4. Select number e to be relative prime to $\Phi(m)$, let $e = 3$
5. Compute number d, so that $e * d = 1 \pmod{\Phi(m)}$
   $d = 27$, because $27 * 3 = 81 = 1 \pmod{40}$

For any number x: $(x^3)^{27} = x \pmod{m}$

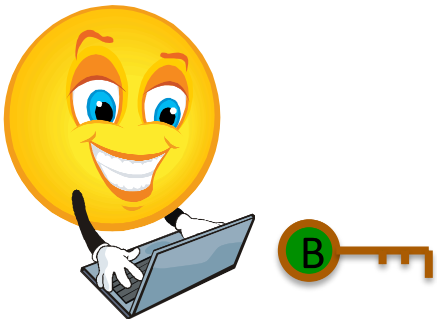
Bob's public key: $m=55$ and $e=3$

Bob's private key: $d=27$

A more detailed example can be found e.g. here

RSA encryption - example

Alice

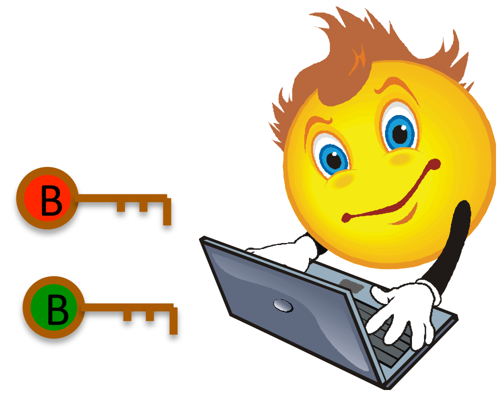


3, 55

Bob

27

3, 55



Alice wishes to send cleartext message m=8 to Bob

8^3=512 which is 17 (modulo 55)

Encrypted message = 17

Alice sends encrypted message 17 to Bob

Bob receives encrypted message 17 from Alice

17^{27}=1 667 711 322 168 688 287 513 535 727 415 473 which is 8 (modulo 55)

The message Alice sent is: 8

RSA signature - example

Bob wishes to sign message 8

8^{27}= 2417851639229258349412352

which is 2 (modulo 55)

Bob sends the message 8 and signature 2 to Alice

Alice receives message 8 and signature 2, and verifies if 2 is a valid signature from Bob on message 8

2^{3}=8 (which is 8 modulo 55)

As 2 is Bob's signature for 8, so the signature is valid.

RSA caveats

- Exponentiation is never performed the previous, naïve way
  - computing modulo after each multiplication
  - square and multiply algorithm – a lot more efficient
  - further acceleration via p and q (based on Chinese Remainder Theorem)
- In certain scenarios, there are efficient attacks, e.g.:
  - very small public exponent (e) values
  - multiple users using the same modulus (m)
  - …

Security of RSA

- The attacker knows
  - the public key (e, m)
  - the encrypted / signed message
- The attacker may choose to
  - factorize m
  - guess the private key
  - guess the decrypted message / signature
  - …
- Factoring integers is believed to be a hard problem
  - it is believed that no polynomial time algorithm exists
- Computing d from (e, m) is equivalent to factoring m
- Computing the message from the ciphertext may not be equivalent to factoring m

Elliptic Curve Cryptography (ECC)

What is an elliptic curve?

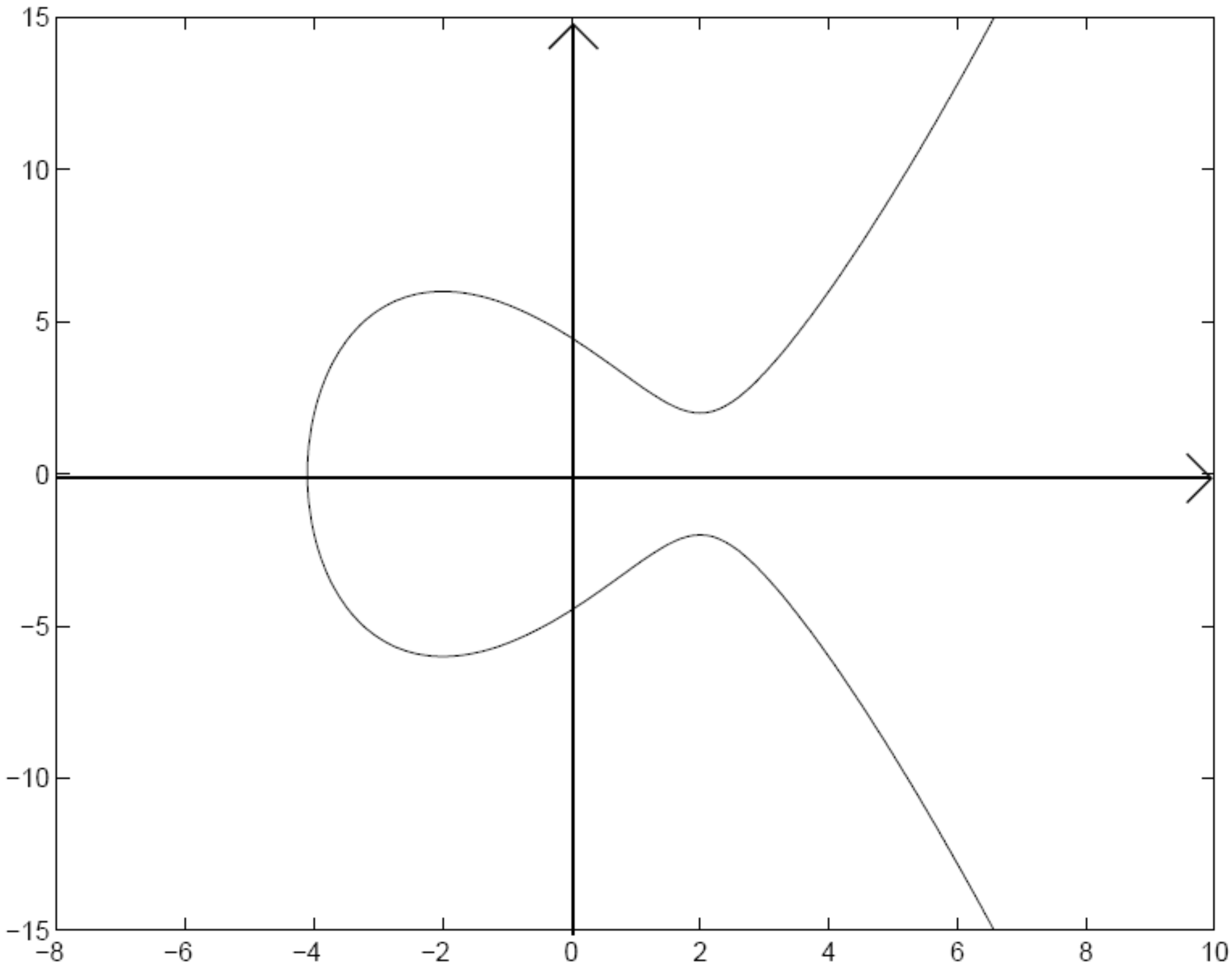
- An elliptic curve consists of points (x,y) that satisfy the below equation:

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

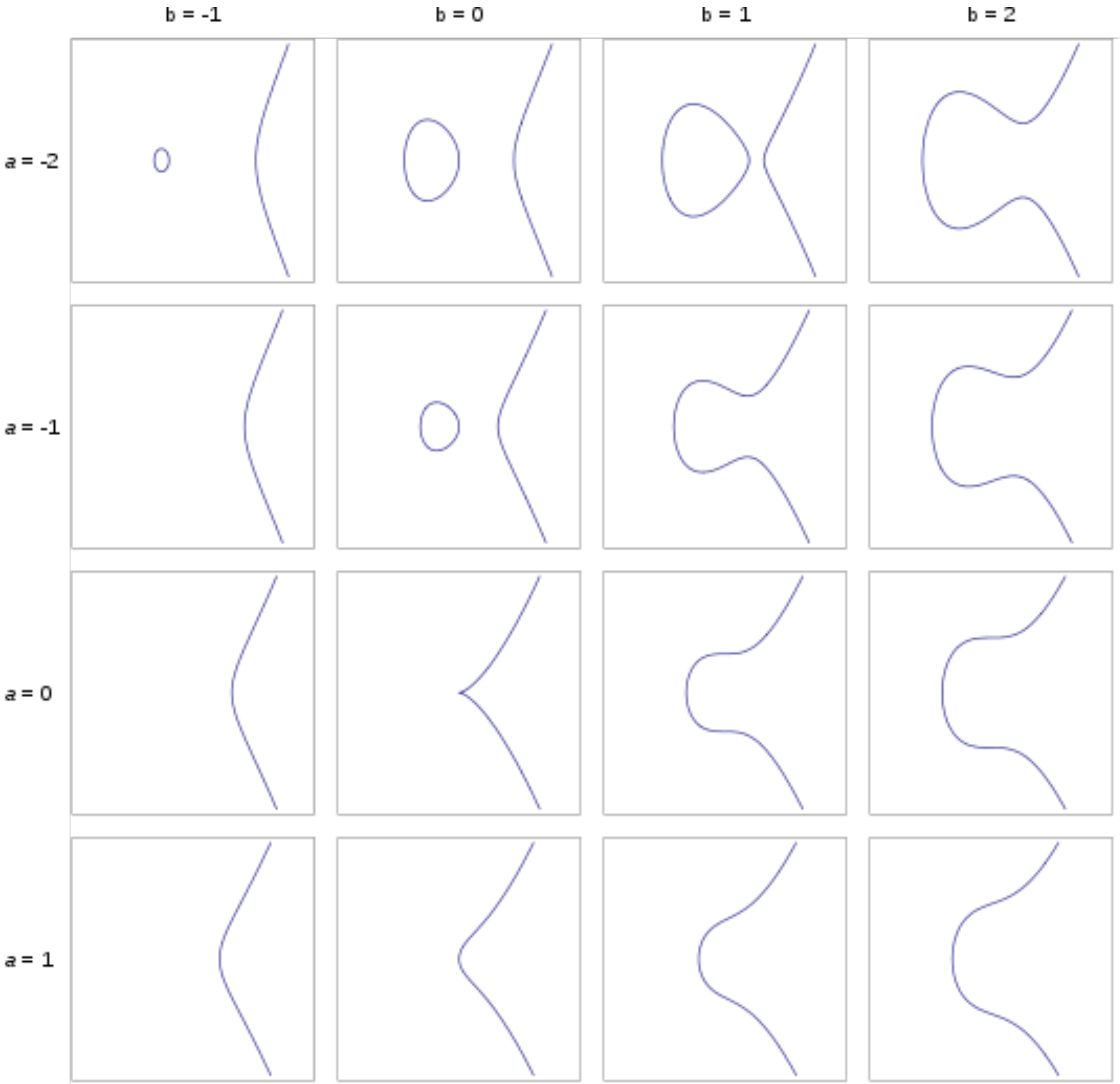
- where constants a, b, c, d, e and variables x, y are elements of field **F**
- Curves over real numbers (where **F**=**R**) can be depicted as graphical curves

- In cryptography, elliptic curves can be used to define mathematical problems that can be used as a basis for public key cryptosystems

An elliptic curve above real numbers (R)



More elliptic curves over real numbers (R)



For real numbers, the equation can be simplified to:

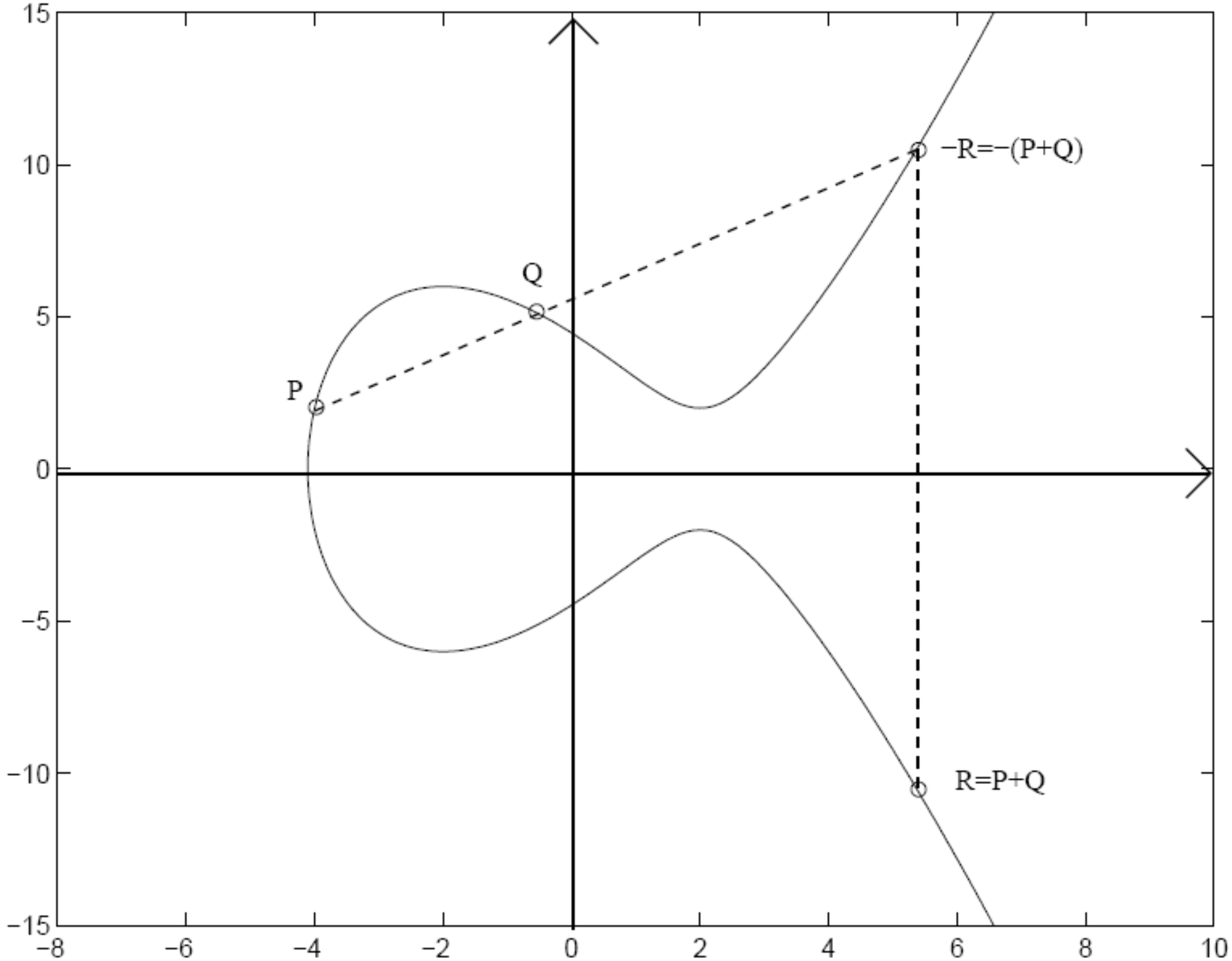
$$y^2 = x^3 + ax + b$$

We can define operations

- We can define operations between points of the curve…
- Why?
- Why not?

Adding points P and Q of the curve

geometrical definition



Adding points P and Q of the curve

algebraic definition – a more general definition

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

for curve $y^2 = x^3 + ax + b$.

The coordinates of R can be obtained as follows:

$$x_3 = s^2 - x_1 - x_2$$

$$y_3 = s(x_1 - x_3) - y_1$$

where s is the 'slope' of the curve.

If $P \neq Q$ then $s = (y_2 - y_1)/(x_2 - x_1)$

If $P = Q$ then $(3x_1^2 + a)/2y_1$

If $Q = -P$ then $P + Q = O$, where O is a point of infinity.

Multiplying a point with an integer

- We can define another operation over the points of the curve: multiplying a point with an integer
- Multiplication with an integer – adding the point multiple times to itself
- For example:
  5*P = P + P + P + P + P

Elliptic Curve Discrete Logarithm Problem

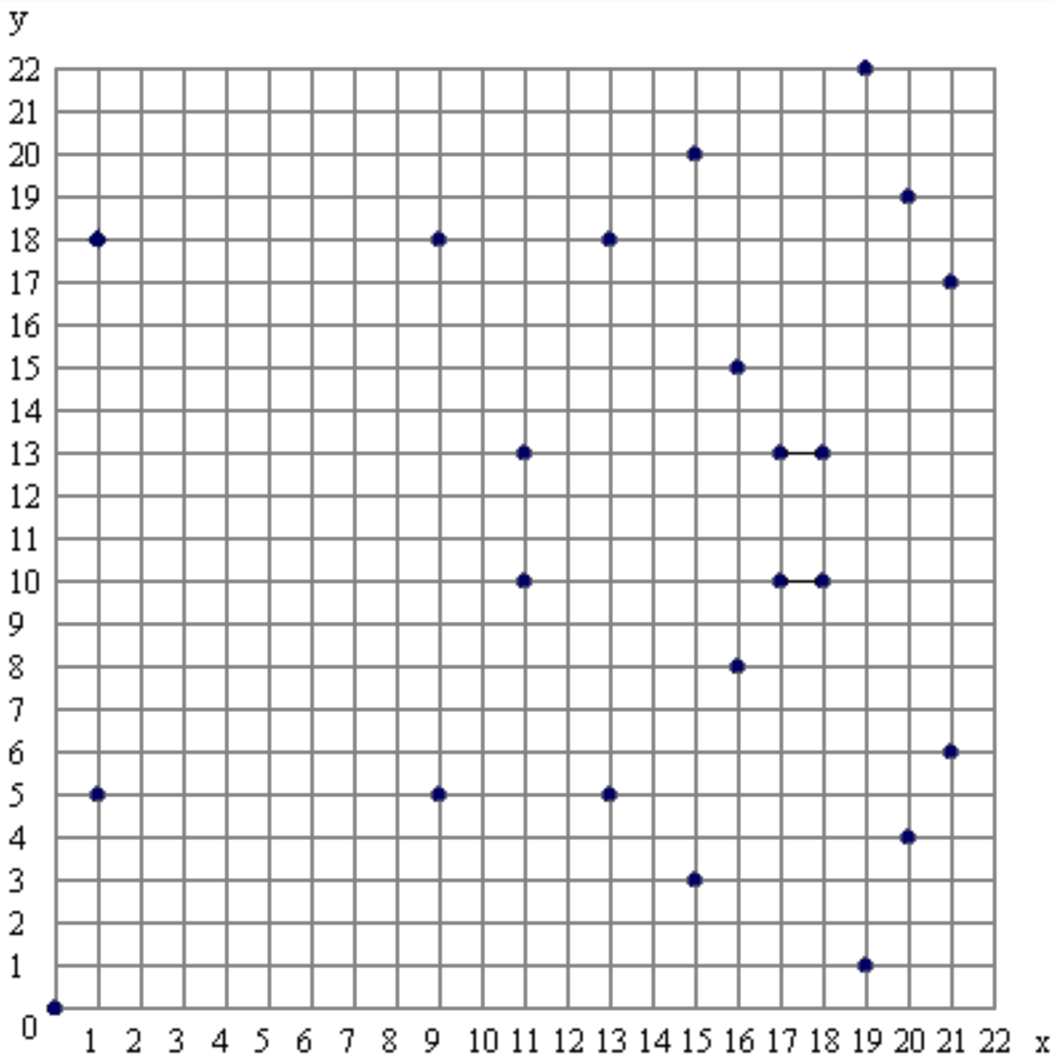
- We define the following operations over elliptic curves:
  - addition of two points of the curve
  - multiplication of a point with an integer
- If Q is a point of the curve and k is an integer, then
  - based on Q and k*Q
  - computing k

  is the Elliptic Curve Discrete Logarithm Problem (ECDLP)
- We look for cases when the ECDLP is a 'hard' problem, i.e. where no efficient algorithm is known
- This depends on the field, and also depends on the actual curve

Over which field?

- Infinite fields are not useful in cryptography due to e.g. rounding and inaccuracy problems.
  - Note: The field of real numbers (**R**) is never used in cryptography, so graphical representations of curves are illustration only.
- GF(p) – the field of integers modulo p, where p is prime; the definition of + is same as the one for real numbers
- GF(2^m) – elements of this field are binary vectors of length m, they can also be represented as polynomials of the mth power; as the characteristic of this field is 2, formulae of the definition of + are slightly different

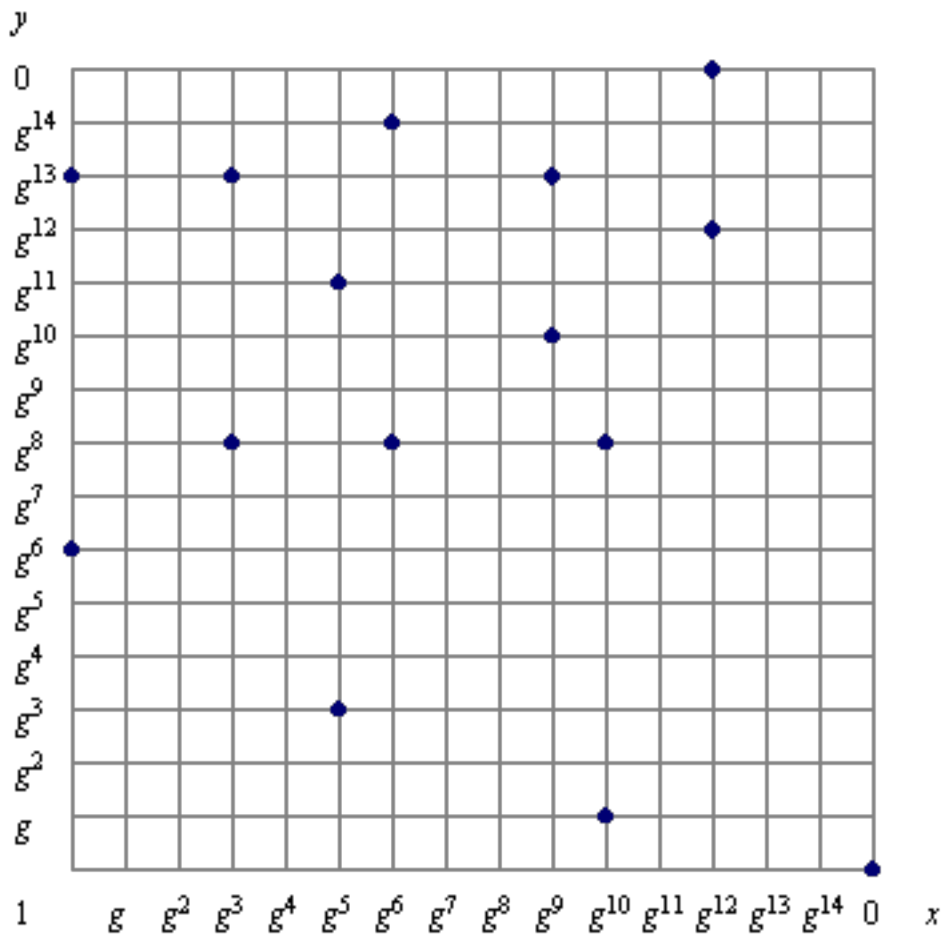
Example curve over GF(p)



Elliptic curve equation: $y^2 = x^3 + x$ over F_{23}

Source: https://www.certicom.com/ecc-tutorial

Example curve over GF(2^m)



$$y^2 + xy = x^3 + g^4x^2 + 1 \text{ over } F_{2^4}$$

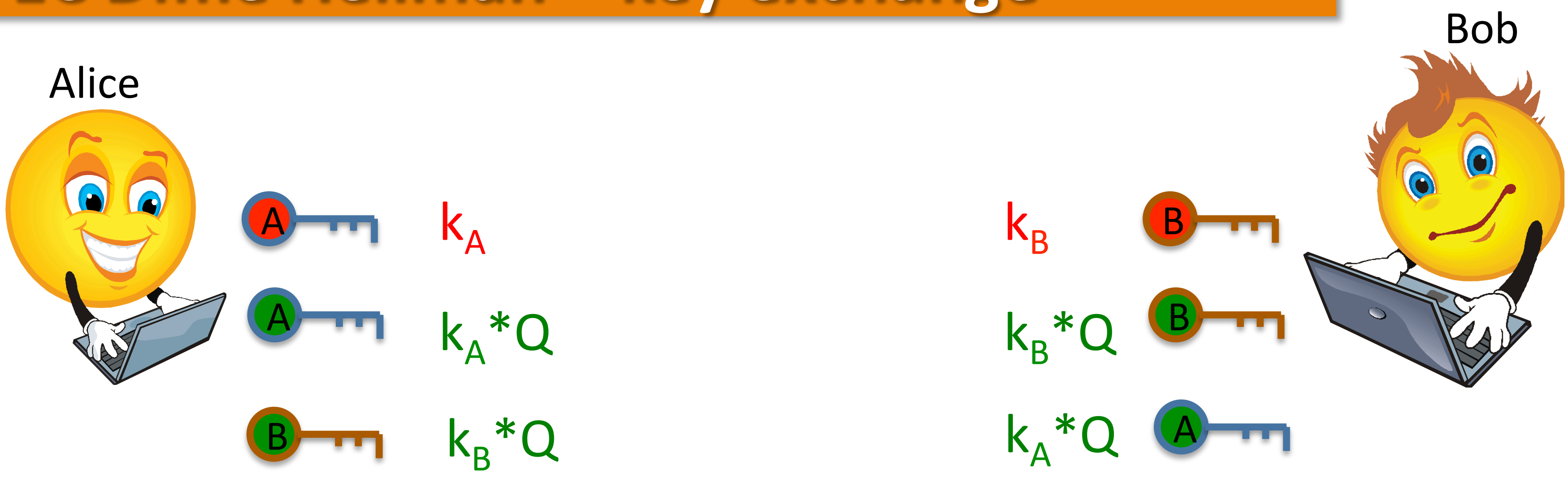
Source: https://www.certicom.com/ecc-tutorial

ECC key generation

- The curve is usually a system-wide parameter; there are recommended curves with good properties
  - NIST curves (US) from nist.gov
  - Brainpool curves (EU) from ecc-brainpool.org

- Q is a base point of a curve, another system-wide parameter
- The private key of user U is k_u, a random integer
- The public key of user U is k_u*Q, a point of the curve

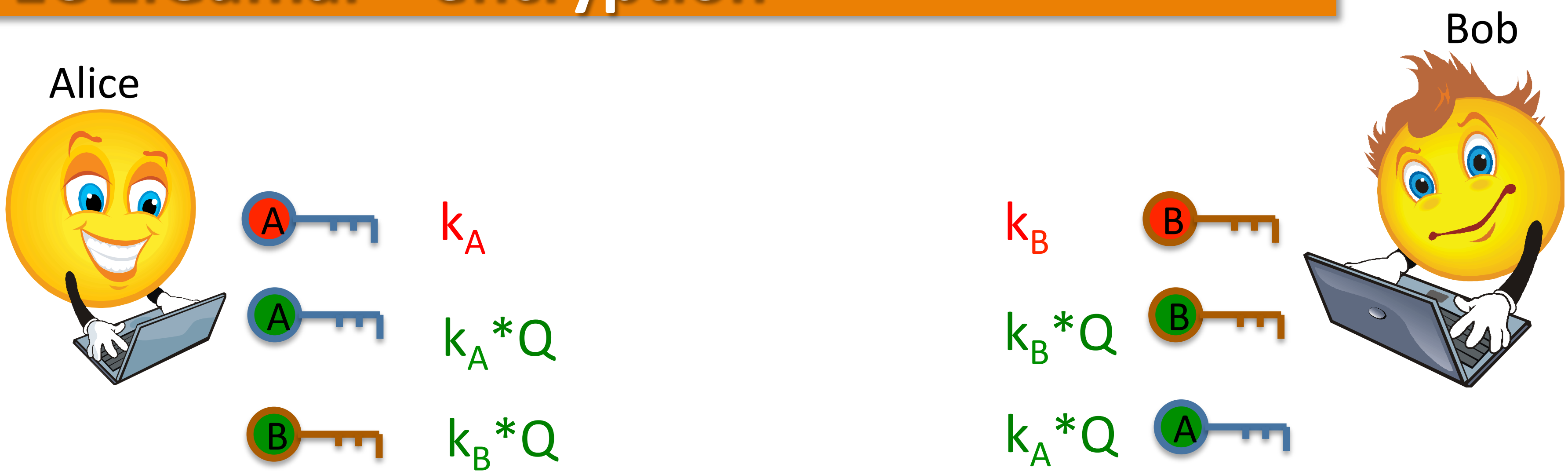
EC Diffie Hellman – key exchange



1. A → B: k_A*Q
2. B → A: k_B*Q
3. Alice computes: $k_B*Q * k_A = k_A*k_B*Q$
   Bob computes: $k_A*Q * k_B = k_A*k_B*Q$

Thus obtain both parties shared secret k_A*k_B*Q that can be used as a (basis for a) symmetric key.

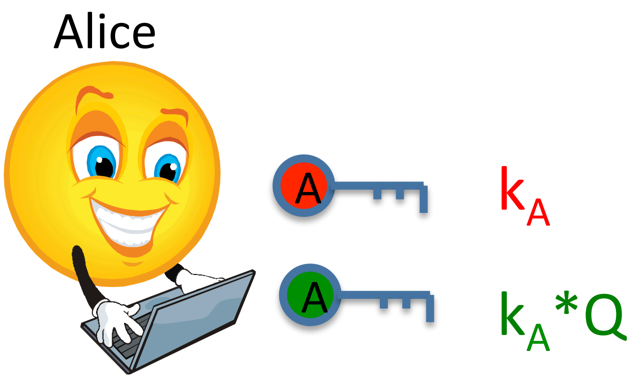
EC ElGamal – encryption



Alice sends message m, represented as point M of the curve.

1. Alice chooses a fresh random number r
2. Alice sends the encrypted message:
   A → B: $r*Q$, $M + r*k_B*Q$
3. Bob decrypts the message by computing k_B*r*Q and
   $M + r*k_B*Q - k_B*r*Q = M$

EC DSA – digital signature (signing)



Alice signs message m:

1. Computes e = h(m) modulo n
   where h is a hash function
2. Generates random number t
   where $t \in [1, n-1]$
3. Computes r = (t*Q)[x] (modulo n)
   where (t*Q)[x] stands for the x coordinate of point t*Q
4. Computes $s = t^{-1}*(e + r*k_A)$ (modulo n)

Alice's signature on message m is r, s:

$r, s = \quad (t*Q)[x], \qquad t^{-1}*(e + r*k_A)$

EC DSA – digital signature (verification)

Bob verifies if

r, s = (t*Q)[x], $t^{-1}*(e + r*k_A)$

is Alice's signature on message m:

1. Bob also computes e = h(m) modulo n
2. Computes $w = s^{-1}$ (modulo n)
3. Computes $u_1 = (e*w)$ and $u_2 = r*w$ (modulo n)
4. Computes point $(x_1, y_1) = u_1*Q + u_2* k_A*Q$
   which is $(x_1, y_1) = Q*(u_1 + u_2* k_A)$
5. Since $s = t^{-1}*(e + r*k_A)$,
   $t = s^{-1}*(e + r*k_A) = w*(e + r*k_A) = (u_1 + u_2* k_A)$
   and thus $(x_1, y_1) = t*Q$
6. The signature is valid iff r is the x coordinate of the above t*Q

Why ECC?

- Provides security with significantly shorter keys than RSA
  - 1024-bit RSA ~ 160-bit ECC
  - 2048-bit RSA ~ 224-bit ECC
- Note that an exact comparison is very hard to be made
  - IFP (RSA) – since the ancient Greek
  - ECDLP (ECC) – since 1985 (Koblitz, Miller)
- ECC has shorter keys but more complex operations, still ECC is often considered faster

- NSA Suite B cryptography → ECC

RSA and ECC

- RSA is fully symmetric
  - public and private keys can be interchangeable (if e was not a fixed value, it could also be made secret)
  - signing and decryption are the same operation

  These are specific to RSA

- The shown ECC algorithms for signing (ECDSA) and encryption (EC ElGamal) need fresh random value
  - in practice, RSA encryption is (or should be) randomized too

Summary

- In public key cryptography, the key has two parts: one part can be used for encryption / signature verification only, this can be made public, the other part is used for decryption / signing, this must be kept private
- Public key cryptography allows ‘signatures’ that can be verified by anyone using the public key
- The public key and the private key needs to be interlinked, but there must not be an efficient way for computing the private key from the public part
- Public key cryptosystems are based on mathematical problems with the above properties
  - RSA: Integer Factorization Problem
  - ECC: Elliptic Curve Discrete Logarithm Problem