

IT Security



Chapter 2: Encryption (Part 2)

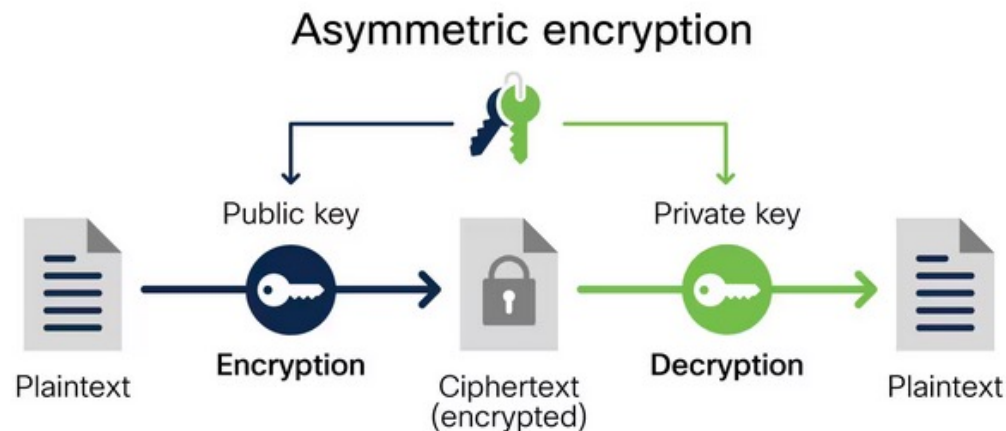
- ▶ Asymmetric encryption
- ▶ Practical aspects of encryption
- ▶ Random numbers
- ▶ Key Management





Asymmetric encryption

- ▶ Based on the use of key pairs (private and public key)
- ▶ There are fewer methods because they are more difficult to construct (they are based on problems of algorithmic number theory)
- ▶ Disadvantage: High computing effort
- ▶ Advantage: Key exchange no problem
- ▶ Examples: RSA, ECIES (Eliptic Curve), DLIES (Discrete Logarithm)

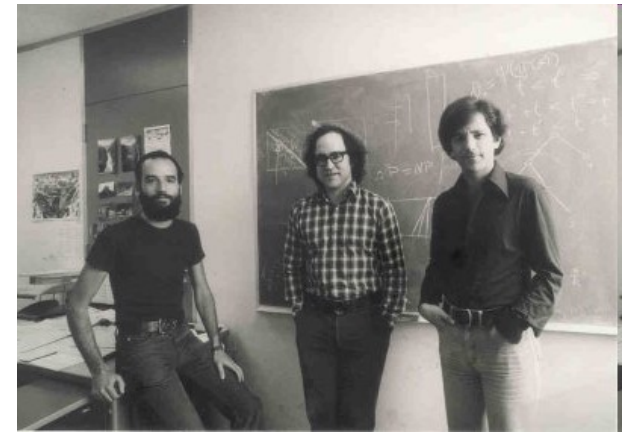


Source: <https://www.cisco.com/c/en/us/products/security/encryption-explained.html#~q-a>



RSA (Rivest-Shamir-Adleman)

- ▶ is an asymmetric algorithm
- ▶ it is easy to understand and implement, popular, license-free
- ▶ To find the secret key from the public one, the prime factors of a number n must be found.
- ▶ Attack: Factoring large numbers is very difficult.
- ▶ key lengths of 2000 or 3000 are currently required
- ▶ The implementations are much slower than AES.





RSA Algorithmus

1. choose 2 large ($> 2^{512}$) prime numbers p and q
2. calculate RSA module $n=p*q$
3. select $e < n$ with $\text{GCD}(e, (p-1)(q-1))=1$
4. select d so that $ed \bmod (p-1)(q-1) = 1$
5. public key is (e,n) , private key is (d,n)

Encrypt: $c = m^e \bmod n$

Decrypt: $m = c^d \bmod n$

▶ Elliptic Curve Cryptography

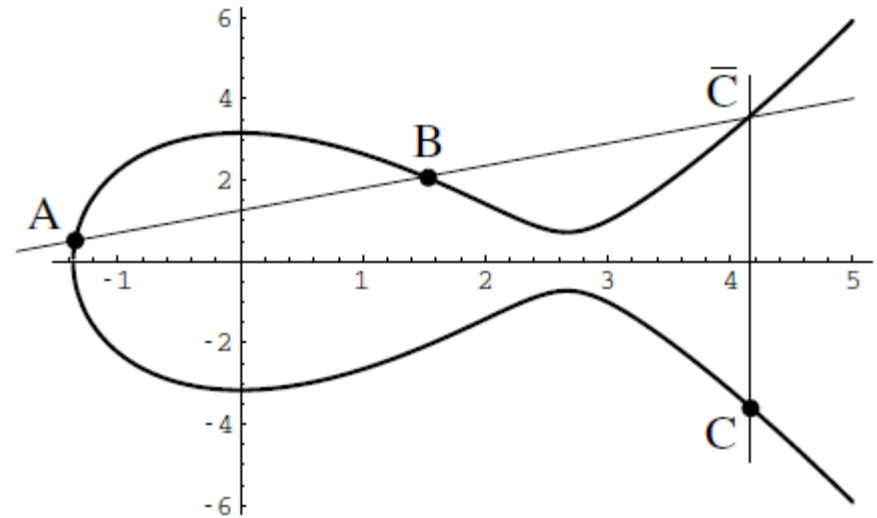
▶ Example for a curve equation:
 $y^2 = x^3 - 4x^2 + 10$

- ▶ Advantages of ECC:
- ▶ shorter key lengths as RSA
 - ▶ this makes it faster
 - ▶ safe at quantum computers

▶ Procedure: several passes with straight lines through two points and reflection of the third intersection

▶ Public Key: start and end point, Private Key: number of rounds

▶ Video for illustration



Source:: Angewandte Kryptographie, W.Ertl, Hanser

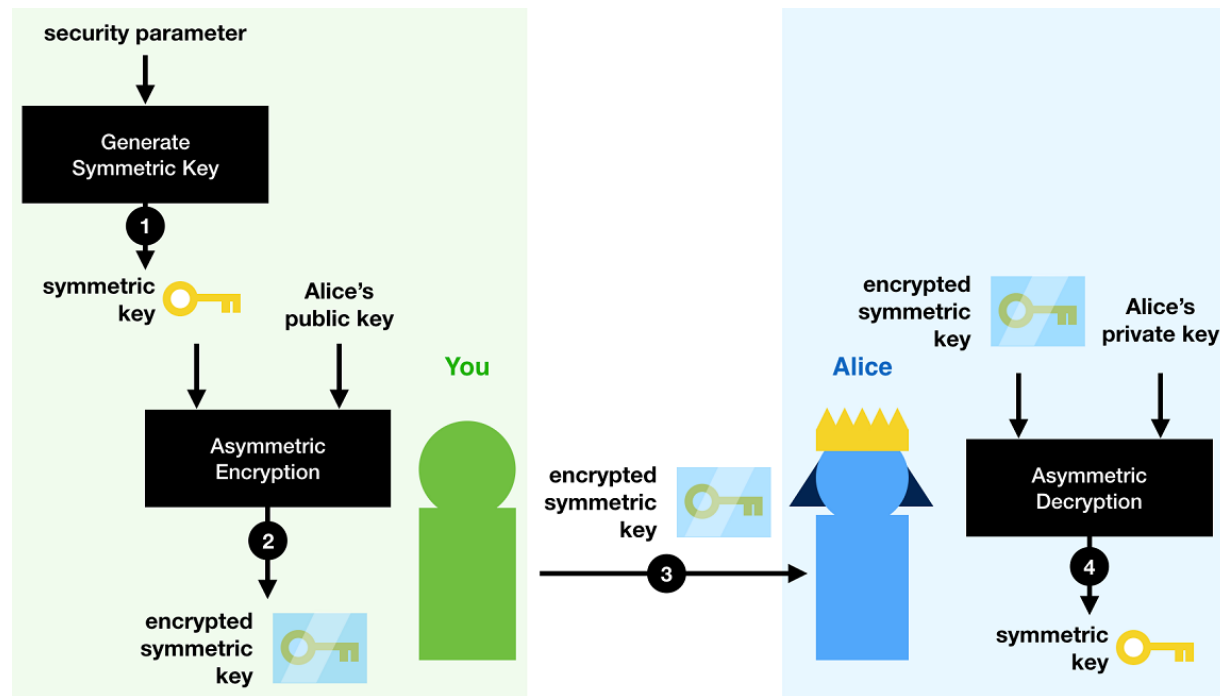


<https://www.youtube.com/watch?v=dCvB-mhkT0w>



Hybrid Encryption

- ▶ Coupling symmetric and asymmetric encryption
- ▶ The message is first symmetrically encrypted (session key) and then the session key is encrypted asymmetrically
- ▶ Advantage: High speed of encryption and secure key exchange

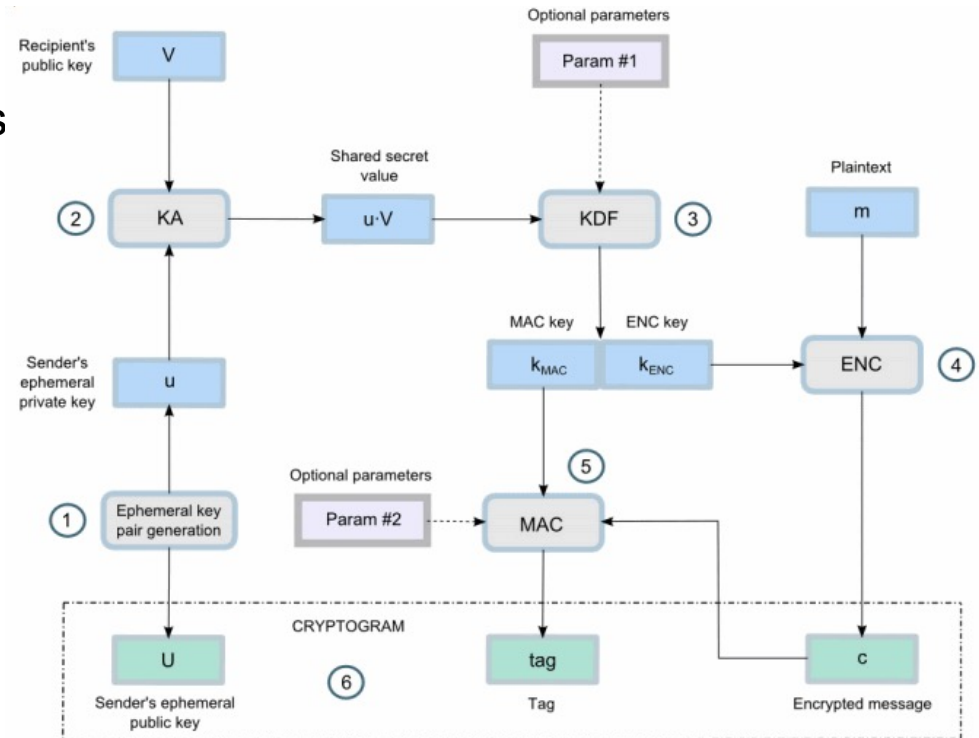


Source: Real-World Cryptography, David Wong, Manning Publications (2021)



ECIES Elliptic Curve Integrated Encryption Scheme

- ▶ **Hybrid encryption method** that combines ECC asymmetric cryptography with symmetric cipher and MAC
- ▶ ECIES is not a concrete algorithm but a framework
- ▶ You can use different algorithms (AES-CTR, AES-GCM, ChaCha20-Poly1305, ...)



KA Key Agreement (e.g. ECDH)
 KDF Key Derivation Function (e.g. cryptographic hash)
 ENC Encryption

- ▶ Further details can be found at

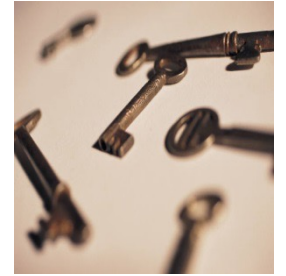
<https://medium.com/asecuritysite-when-bob-met-alice/go-public-and-symmetric-key-the-best-of-both-worlds-ecies-180f71eebf59>

https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=12





Practical aspects of encryption



- ▶ How do I build cryptography into my applications?
- ▶ In which format do I store encrypted data?
- ▶ What attacks are there on encryption?
- ▶ What key lengths and procedures are currently required?
- ▶ How do I create suitable keys?
- ▶ How do I get cryptographically secure random numbers?
- ▶ What do I have to consider when managing keys?
- ▶ How do I exchange keys securely?



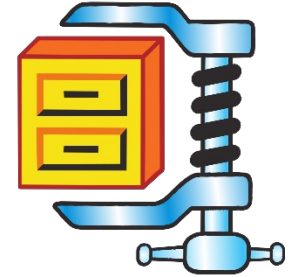
How do I build cryptography into my applications?

- ▶ Never do "**Security by obscurity**":
 - ▶ Security by obscurity: System is a kind of black box, parts of the process or algorithm must be kept secret.
- ▶ Better apply **Open Design**:
 - ▶ "The security of a mechanism should not depend on the secrecy of its design or implementation."
OWASP Secure Design Principles <https://github.com/OWASP/DevGuide/blob/master/02-Design/01-Principles%20of%20Security%20Engineering.md>
- ▶ **Kerkhoffs principle**:
 - ▶ The security of a cryptographic method may depend solely on the key used
- ▶ Do not use your own cryptography !!!
 - ▶ But use proven crypto libraries



What else do I have to consider?

- ▶ Compression – Encryption – Error Detection
 - ▶ the right order is important
 - ▶ **compression** makes sense because
 - ▶ encryption becomes faster when compression is fast
 - ▶ cryptanalysis (based on the exploitation of redundancy in plain text) is made more difficult
 - ▶ an additional signature or checksum on encrypted data makes sense by recognizing manipulations of the cipher text
- ▶ When using cryptography in software products, **export restrictions** and **patents** must be respected
 - ▶ cryptography falls under the Weapons Act
 - ▶ Bureau of Industry and Security (<https://www.bis.doc.gov/>) regulates in USA export and import of technologies that use cryptography





A format for storing encryption: XML Encryption Standard

- ▶ W3C Recommendation (2013) <https://www.w3.org/TR/xmlenc-core/>
 - ▶ Describes the XML syntax for storing encrypted documents in XML.
- ▶ There are the following encryption options
 - ▶ encryption of the entire XML document
 - ▶ encryption of a single element and its sub-elements
 - ▶ encryption of the contents of an XML element
 - ▶ multi-recipient encryption
- ▶ There are the following elements for the xml structure
 - ▶ **EncryptedData**: the enveloping tag for encryption
 - ▶ **EncryptionMethod**: Encryption method
 - ▶ **CipherData**: provides encrypted data directly or indirectly by reference (**CipherValue**, **CipherReference**)
- ▶ Advantage of XML encryption: supported by many libraries



Sample for XML Encryption: XML document with payment info

```
<PaymentInfo xmlns='http://example.org/paymentv2`'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD`'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Task: Encrypt the **credit card information** for secure transport



XML document hybrid encrypted with public key of recipient

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
    xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod Algorithm=
      'http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
    <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
      </EncryptedKey>
    </KeyInfo>
    <CipherData>
      <CipherValue>ydUNqHkMrD...</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

unencrypted content

processing information

key management hooks

encrypted key for secure transport

encrypted data



Another format for storing encryption: PKCS#7 Enveloped Data

- ▶ used in SMIME
- ▶ can be combined with signatures: *Signed and Enveloped Data*
- ▶ syntax of *Enveloped Data*

We will learn more details in
the chapter Digital Signatures

```
EnvelopedData ::= SEQUENCE {  
    version                Version,  
    recipientInfos          RecipientInfos,  
    encryptedContentInfo    EncryptedContentInfo  
}  
RecipientInfos ::= SET OF RecipientInfo  
EncryptedContentInfo ::= SEQUENCE {  
    contentType            ContentType,  
    contentEncryptionAlgorithm ContentEncryptionAlgorithmIdentifier,  
    encryptedContent [0] IMPLICIT EncryptedContent OPTIONAL  
}  
EncryptedContent ::= OCTET STRING
```



Attacks on encryption, cryptanalysis

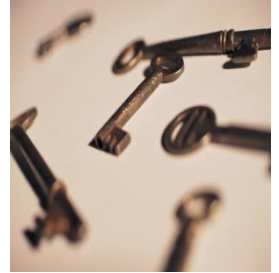


- ▶ Cryptanalysis: Mathematically analytical
 - ▶ Known Ciphertext, Known Plaintext, Chosen Plaintext, Chosen Ciphertext
- ▶ Brute Force
- ▶ Social Engineering
- ▶ Search for vulnerabilities in the overall system in order to obtain keys
- ▶ Side Channel Attack
 - ▶ attack on a cryptographic system that exploits the results of physical measurements on the system (for example, energy consumption, electromagnetic radiation, time consumption of an operation) to gain insight into sensitive data.
- ▶ Fault Attack
 - ▶ attack on a cryptographic system in which the attacker uses or actively causes a faulty execution of a cryptographic operation.
- ▶ Man-in-the-middle attack



What key lengths and procedures are currently required?

- ▶ Key Length – Period of Validity
 - ▶ symmetric methods:
 - ▶ attacks via Brute Force
 - ▶ length at least 256
 - ▶ asymmetric methods:
 - ▶ attacks via mathematical methods
 - ▶ length at least 2000 (RSA), 200 (ECC)
 - ▶ Cryptosystems must be built in such a way that it is possible to change the methods and key lengths.
- ▶ This information must be reviewed regularly
- ▶ An important source is the BSI



Source for currently recommended procedures and key lengths:

https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile&v=6



Key Management



- ▶ Cryptographic methods are only as secure as the keys are protected
- ▶ A cleanly implemented key management is required for this
- ▶ Important Key Management features
 - ▶ generation of keys
 - ▶ storage of keys
 - ▶ secure transmission – exchange of keys
 - ▶ withdrawal of invalid or compromised keys
 - ▶ backup copies of keys
 - ▶ destruction of keys



Key Management: Storage



- ▶ Where do I store keys, credentials, tokens?
 - ▶ in source code? -> never!!
 - ▶ in configuration files, environment variables:
either strict access control or encrypt
 - ▶ Password Based Encryption (**PBE**)
 - ▶ Example: Java Keystore
 - ▶ external keystores
 - ▶ **Vaults/Secret Manager** as a service or in a HSM (Hardware Security Module),
access via API with authentication and authorization and with audit logs
Example: Microsoft Azure Key Vault, AWS Secrets Manager
 - ▶ **Sealed Secrets** in git based on asymmetric crypto
<https://learnk8s.io/kubernetes-secrets-in-git>
- ▶ Chip card and PIN
- ▶ Key splitting (secret splitting, secret sharing: n:m principle)

Split keys on n instances
(persons), m instances are
required to reconstruct keys



Key management: generate, transfer, withdraw/lock

- ▶ Key generation
 - ▶ good random numbers are required
 - ▶ they should be generated in a secure place (e.g. smart card, HSM)
- ▶ Secure transmission - exchange of keys
 - ▶ mostly based on asymmetric cryptography (DH, ECDH, RSA and PKI)
- ▶ If a key is compromised, it must be exchanged
 - ▶ mark key as locked
 - ▶ replace key
 - ▶ re-encrypt all affected data
 - ▶ the actions **lock-exchange-recrypt** are also necessary if a key is no longer secure (encryption procedure insecure, length of key too short)



Key management: save, destroy

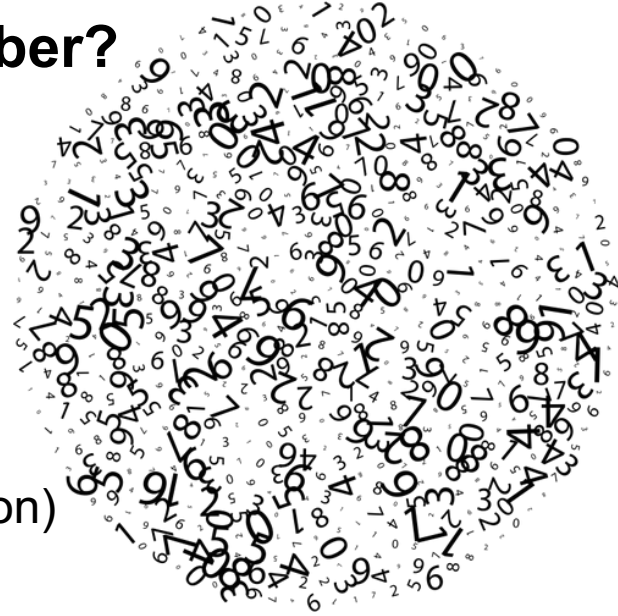


- ▶ Backup copies of keys are usually necessary
 - ▶ for exceptional situations (death, termination of employment, vacation, ...) and loss
 - ▶ Key **Recovery**: deposit for key loss
 - ▶ Key **Escrow**: deposit for law enforcement
 - ▶ Secret Sharing (e.g. Shamir Secret Sharing)
 - ▶ Master Key
 - ▶ ATTENTION: do not make backup copy of private signature keys
- ▶ Key destruction
 - ▶ overwrite files and hard disk memory areas, delete cache and swap areas
 - ▶ hardware may have to be destroyed

otherwise
signatures can be
faked



Randomness: How random is a random number?



- ▶ Initial situation
 - ▶ the quality of the encryption depends on the quality of the keys
 - ▶ the entropy is a measure for the information content of a message (maximum entropy means equal distribution)
 - ▶ many other algorithms need random numbers (e.g. salt for hash)
- ▶ Problem:
 - ▶ computers are deterministic machines, randomness is hard to get
- ▶ Solution:
 - ▶ collect as many sources of entropy as possible that are hard to predict (interrupts, user input, noise from sound card, etc.)

73735	45963	78134	63873
02965	58303	90708	20025
98859	23851	27965	62394
33666	62570	64775	78428
81666	26440	20422	05720
15838	47174	76866	14330
89793	34378	08730	56522
78155	22466	81978	57323
16381	66207	11698	99314
75002	80827	53867	37797
99982	27601	62686	44711
84543	87442	50033	14021
77757	54043	46176	42391
80871	32792	87989	72248
30500	28220	12444	71840



Cryptographic random numbers



- ▶ Properties of a good random number generator
 - ▶ it generates evenly distributed numbers
 - ▶ the numbers cannot be predicted
 - ▶ it has a long and complete cycle
- ▶ Problem: SW-implemented **PRNG's** (Pseudo Random Number Generator) are mostly fast but predictable. (e.g. mathematical random functions)
- ▶ Solutions:
 - ▶ **True RNG's** based on HW (like electronic noise) are expensive and complex.
 - ▶ PRNG mix different random sources (time, internal CPU counters, keystrokes, mouse movement etc.) as initial value and use a progressive function to calculate the random number
 - ▶ Cryptographic libraries offer PRNG:
 - .Net: `System.Security.Cryptography.RNGCryptoServiceProvider`,
 - JCE: `java.security.SecureRandom`
 - ▶ use `/dev/random` (GNU/Linux) or `getrandom(2)` syscall



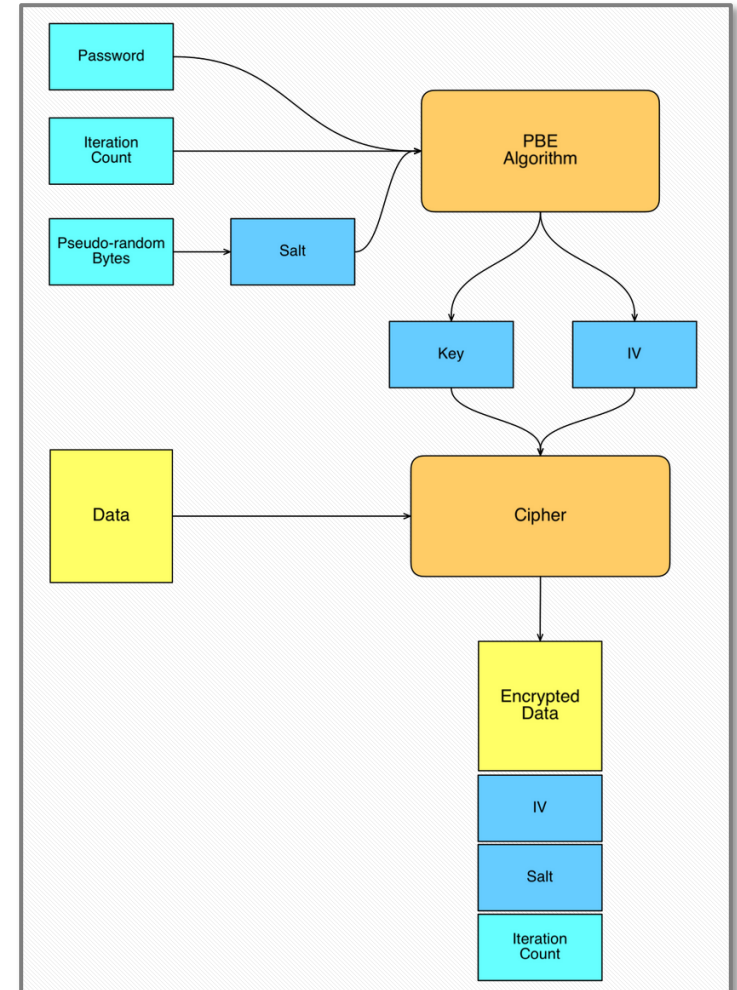
Example

```
private void random() {  
    SecureRandom random = new SecureRandom();  
  
    byte[] data = new byte[256 / 8];  
    random.nextBytes(data);  
    System.out.println(bytesToHex(data));  
  
    // Do not create a new SecureRandom, reuse it  
    byte[] moreData = new byte[256 / 8];  
    random.nextBytes(moreData);  
    System.out.println(bytesToHex(moreData));  
}
```



Password Based Encryption (PBE)

- ▶ PBE allows to generate cryptographic keys from passwords
- ▶ There are standardized methods
 - ▶ **PKCS#5** using **PBKDF2** (Password-Based Key Derivation Function2) a standardized function to derive a key from a password
 - ▶ **PKCS#12** defines a file format that is used to store private keys with the associated certificate in a password-protected manner.
- ▶ **Important:** The password must be good (high entropy) otherwise attacks are possible (e.g. dictionary attack).



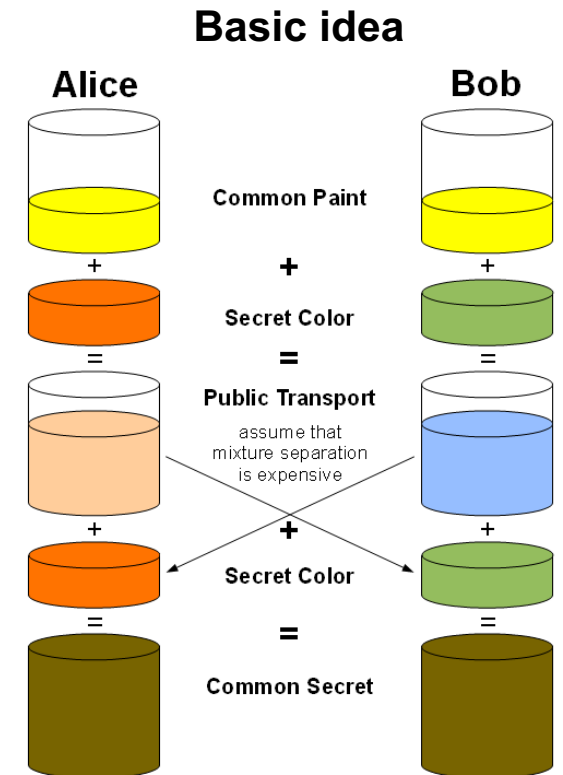
Schematic representation of PBE

Source: <http://www.crypto-it.net/eng/theory/pbe.html>



Diffie-Hellman key exchange method

- ▶ DH is a public key algorithm for the distribution of keys
- ▶ DH is not suitable for encryption
- ▶ Each communication partner calculates the common, secret session key for itself
- ▶ Thus, the key does not have to be transported over the network
- ▶ DH is based on the mathematical problem of the discrete logarithm
- ▶ The recommended key length is 2000 - 3000 bits



https://commons.wikimedia.org/wiki/File:Diffie-Hellman_Key_Exchange.png



Diffie-Hellman key exchange



▶ Protocol flow

- ▶ Alice and Bob agree on a large prime number n and a number g
 - ▶ Alice chooses a large random number x and sends $X = g^x \bmod n$ to Bob
 - ▶ Bob chooses a large random number y and sends $Y = g^y \bmod n$ to Alice
 - ▶ Alice calculates $k = Y^x \bmod n$
 - ▶ Bob calculates $k' = X^y \bmod n$
 - ▶ At the end $k' = k$
-
- ▶ **Public key:** n, g, X, Y
 - ▶ **Private key:** x, y, k
 - ▶ **Key length:** length of n in bit



Elliptic Curve Diffie-Hellman ECDH

- ▶ the security is based on the difficulty of the Diffie-Hellman problem in elliptic curves
 - ▶ The key lengths is much shorter than for classical DH

p prime number
a,b curve parameter
P base point on curve
q order of P
i cofactor

▶ Protocol flow

- ▶ choose cryptographically strong EC system parameters (p, a, b, P, q, i), key length q should be at least 250 bits
- ▶ exchange system parameters and all following steps authentically
- ▶ A chooses equally distributed a random value $x \in \{1, \dots, q - 1\}$ and sends $Q_A = x \cdot P$ to B
- ▶ B chooses equally distributed random value $y \in \{1, \dots, q - 1\}$ and sends $Q_B = y \cdot P$ to A
- ▶ A calculates $x \cdot Q_B = xy \cdot P$
- ▶ B calculates $y \cdot Q_A = xy \cdot P$

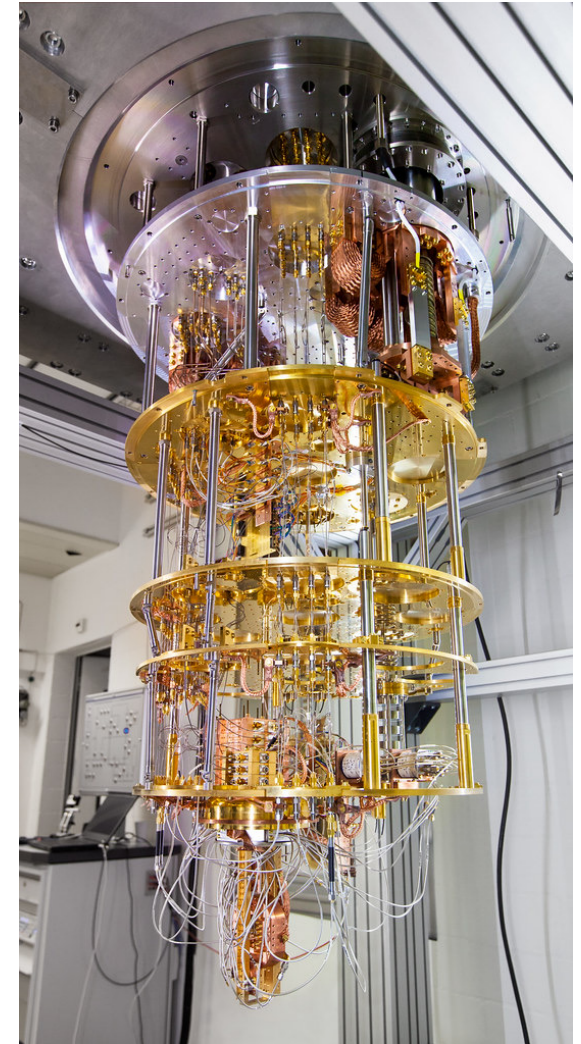
▶ Video for illustration



<https://www.youtube.com/watch?v=yDXiDOJgxmg>



▶ Quantum computing threatens cryptography

- ▶ Quantum computers endanger asymmetric procedures
- ▶ Symmetric methods are not affected so much (just increase key length to be safe)
- ▶ Quantum computers are only theory so far
 - ▶ but they will come in any case
 - ▶ secret services/organizations can collect data today and decrypt it in the future
 - ▶ the replacement of today's cryptography and infrastructure is time consuming





Quantum secure cryptography

- ▶ Therefore, there are already approaches for quantum secure cryptography today
 - ▶ **PQC Post Quantum Cryptography**: mathematical problems that cannot be solved by QC (e.g. FrodoKEM, Classic McEliece)
 - ▶ **QKD Quantum Key Distribution**: secure key distribution (e.g. NewHope key exchange)
- ▶ Competition of NIST
 - ▶ NIST seeks quantum-resistant public-key algorithms
<https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>
- ▶ Videos about Quantum Cryptography
 - ▶ What is Post Quantum Cryptography
 <https://www.youtube.com/watch?v=zw1KHLOOIA8>
 - ▶ Mathematical explanation how quantum computers can crack current algorithms
 <https://www.youtube.com/watch?v=lvTqbM5Dq4Q&t=25s>



Conclusion

- ▶ Learning: Crypto is hard
 - ▶ applies not only to the algorithms, but also to their use
 - ▶ applies also to the chapter on hash, MAC, signatures
- ▶ Take the right library
 - ▶ Tink von Google <https://github.com/google/tink>
 - ▶ NaCl (<https://nacl.cr.yp.to/>) or Sodium (<https://libsodium.gitbook.io/doc/>)
 - ▶ Lazysodium für Java
<https://github.com/terl/lazysodium-java>
- ▶ Literature
 - ▶ <https://cryptobook.nakov.com/>
 - ▶ <https://www.garykessler.net/library/crypto.html>



Summary for encryption



- ▶ Symmetric encryption is fast.
- ▶ Hybrid encryption enables fast encryption with secure key exchange.
- ▶ The best encryption algorithm is of no use with poor key management.
- ▶ It is important to use the known methods correctly and also to pay attention to the details (random numbers, operating modes, initialization vector, checksums, ...).