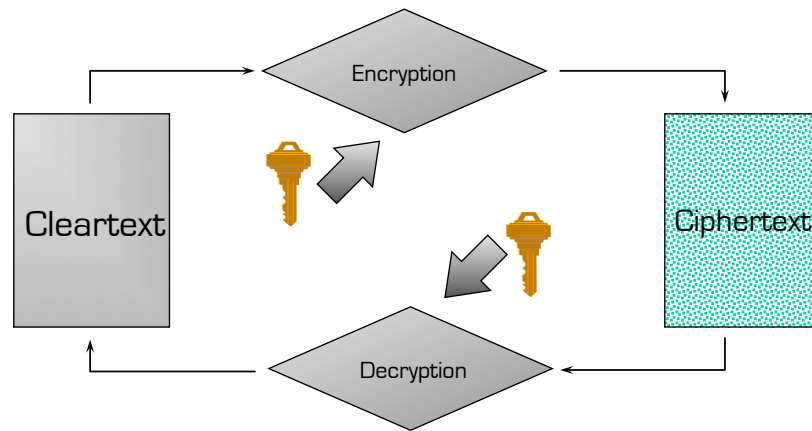


The Process



Giovanni Vigna, Advanced Topics in Security

3

Cryptanalysis

- Cryptanalysis attempts to discover the key or the plaintext of an encrypted message
- Types of attack:
 - Ciphertext only
 - Given: $C_1 = EK(P_1)$, $C_2 = EK(P_2)$, ..., $C_i = EK(P_i)$
 - Obtain: either P_1, P_2, \dots, P_i or K
 - Known plaintext
 - Given: $P_1, C_1 = EK(P_1)$, $P_2, C_2 = EK(P_2)$, ..., $P_i, C_i = EK(P_i)$
 - Obtain: either K or an algorithm to obtain P_{i+1} , from $C_{i+1} = EK(P_{i+1})$
 - Chosen plaintext
 - Given: $P_1, C_1 = EK(P_1)$, $P_2, C_2 = EK(P_2)$, ..., $P_i, C_i = EK(P_i)$ where the attacker chooses P_1, P_2, \dots, P_i
 - Obtain: either K or an algorithm to obtain P_{i+1} , from $C_{i+1} = EK(P_{i+1})$

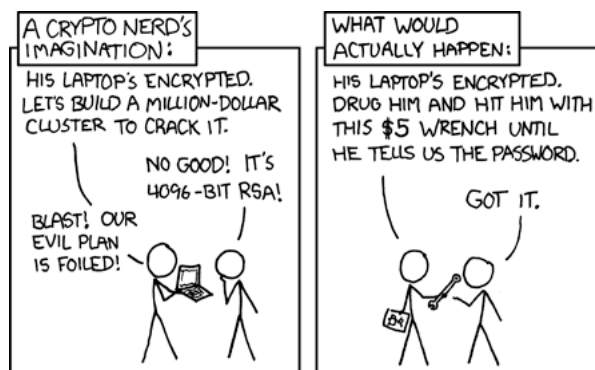
Giovanni Vigna, Advanced Topics in Security

4

Security of a Cryptosystem

- The only perfectly secure algorithm is the one-time pad
- Crypto algorithms can be computationally secure
 - The cost of breaking a cipher exceeds the value of the encrypted information
 - The time required to break the cipher exceeds the useful lifetime of the information
- Cost and time are difficult to estimate
- Exhaustive key space search (brute-force)
 - If trying all possible keys requires ten billion years with the most powerful known computer, the algorithm can be considered secure
- If there are no backdoors...

Security of a Cryptosystem... in the real world



Crypto Algorithms

- Symmetric algorithms (conventional, single-key, secret-key)
 - The encryption/ decryption keys are the same (K)
- Public-key algorithms (asymmetric)
 - There are two different keys (K1, K2)
 - It is not possible (or computationally infeasible) to calculate K1 given K2 or vice versa
 - Whatever is encrypted with one key, can only be decrypted with the other key
- One-way hash functions
 - Takes a variable length input and produces a fixed length output
 - Transformation is irreversible and as similar as possible to a random function
 - Unfeasible to change the message without changing the hash
 - Unfeasible to pre-compute the whole hash space (pre-image resistance)
 - Unfeasible to find a (previously unknown) message that generates a specific given hash (second pre-image resistance)
 - Unfeasible to find two messages that generate the same hash (collision resistance)
 - Transformation can be parameterized using a secret key (K)

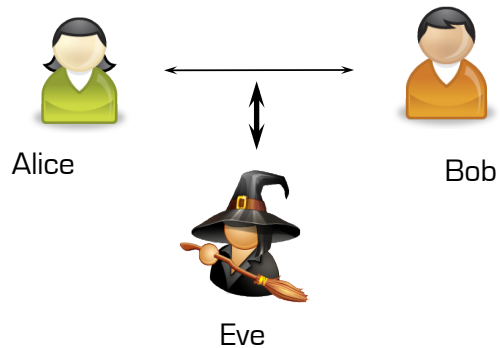
Examples

- Symmetric algorithms
 - DES
 - AES
 - RC4
 - IDEA
- Public-key algorithms
 - Diffie-Hellman
 - RSA
 - Elliptic curve
- One-way hash functions
 - MD5
 - SHA-256
- Random number generators
 - Random number generators (use physical phenomena)
 - Pseudorandom number generators (use initial seed and an algorithm)

Secure Communication

Secure communication means:

- Confidentiality
- Authentication
- Integrity
- Non-repudiation



Communication Using Symmetric Cryptography

- Alice and Bob agree on a cryptosystem
- Alice and Bob agree on a [secret] key K
- Alice encrypts a message using K and she sends it to Bob
$$C = E_K\{M\}$$
- Bob decrypts the message using K
$$M = D_K\{C\}$$

Properties

- Pros
 - *Confidentiality*: Eve cannot access M without knowing K
 - *Authentication* (weak): only who knows K can participate in the communication
 - *Integrity*: if the message C is modified it will decrypt to gibberish
- Cons
 - Key distribution is critical and requires out-of-band communication
 - If key is compromised Eve can impersonate both Alice and Bob
 - The number of keys increases with the number of parties
 - Cannot be used to prove that the message was sent by specifically one of the involved parties

Communication with a Key Distribution Center

- Alice shares a secret key K_a with the KDC
- Bob shares a (different) secret key K_b with the same KDC
- Alice tells the KDC that she wants to talk to Bob
- The KDC generates a temporary secret key (session key) K_t
- The KDC sends $E_{K_a}(K_t)$ to Alice and $E_{K_b}(K_t)$ to Bob
- Alice and Bob communicate using K_t

Properties

- Pros
 - Limited number of keys
 - Kt validity is limited to a session
- Cons
 - The KDC is a single point of failure. If compromised, the whole system is compromised
 - Requires constant availability of the KDC

Communication Using Asymmetric Cryptography

- Alice and Bob agree on a public-key cryptosystem
- Bob sends his public key to Alice
- Alice encrypts a message with Bob's public key and sends it
- Bob decrypts the message using his private key

Properties

- Pros
 - *Key distribution*: Public keys can be sent using in-band channels
 - *Confidentiality*: Eve must know Bob's private key to decrypt the message
 - *Integrity*: any modification of the message would be revealed when decrypting
- Cons
 - No authentication: anybody can send a message to Bob pretending to be Alice
 - Alice can deny to have sent the message

(Better) Communication Using Asymmetric Cryptography

- Alice and Bob agree on a public-key cryptosystem
- Alice and Bob exchange their public keys
- Alice encrypts a message with her private key and then with Bob's public key
- Alice sends the message
- Bob decrypts the message with his private key and then with Alice's public key

Properties

- Pros
 - *Confidentiality*: Eve needs Bob's private key to decrypt the message
 - *Integrity*: message tampering is detected during the decryption process
 - *Authentication*: Alice is the only one who can encrypt with her private key
NOTE: this process is called signing
 - *Non repudiation*: Bob can prove to a third party that Alice is the originator of the message
- Cons
 - Vulnerable to main-in-the-middle attack

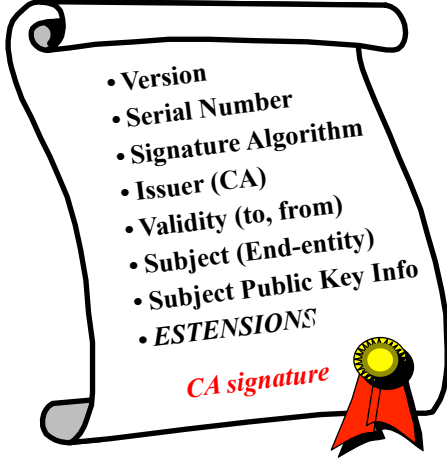
Man-in-the-middle Attack

- Alice sends to Bob her public key
- Eve intercepts the message and sends his own public key to Bob
- Bob sends to Alice his public key
- Eve intercepts the message and sends his own public key to Alice
- Alice sends to Bob a message encrypted with Eve's public key thinking she's encrypting with Bob's public key
- Eve intercepts the message, decrypts it with his own secret key, and re-encrypts it with Bob's public key
- Same for messages from Bob to Alice

Public-key Certificates

- Used to associate an *identity* (e.g., Alice and Bob) with a *public key*
- The association is guaranteed by a trusted third party, the *Certification Authority (CA)*
- The CA owns a public key and a private key
- The CA creates a self-signed certificate that is distributed through many channels
- The CA signs certificates containing identity and corresponding public key after having verified the identity of the requester
- Certificates are made available in public databases or exchanged online

X.509 Certificate

- 
- Version
 - Serial Number
 - Signature Algorithm
 - Issuer (CA)
 - Validity (to, from)
 - Subject (End-entity)
 - Subject Public Key Info
 - *EXTENSIONS*

CA signature



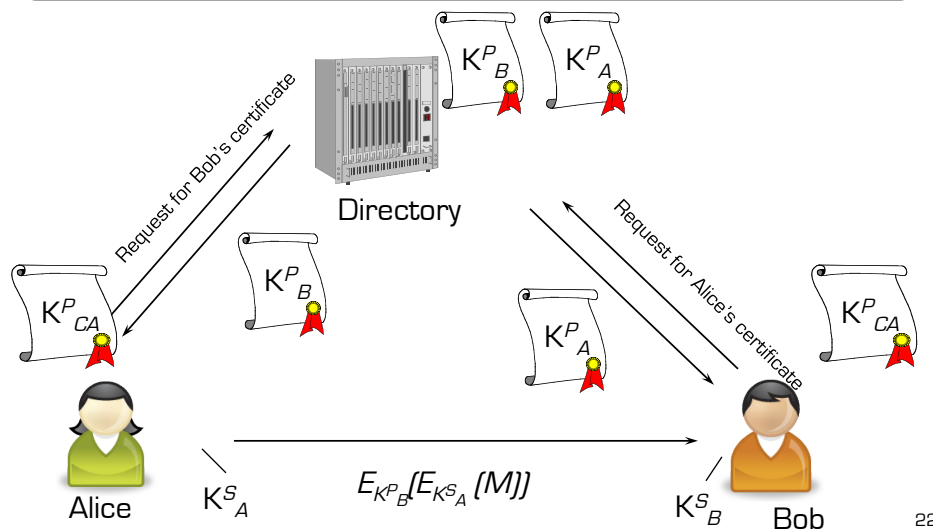
Communication Using Certificates

- Both Alice and Bob have the CA self-signed certificate
- When Alice wants to send a message to Bob
 - She retrieves Bob's certificate from a public database
 - She verifies the CA's signature on Bob's certificate
 - She extracts Bob's public key
 - She uses the Bob's public key and her own secret key to encrypt the message
- When Bob receives the message
 - He retrieves Alice's certificate from a public database
 - He verifies the CA's signature on Alice's certificate
 - He extracts Alice's public key
 - He uses his own private key to decrypt the message and Alice's public key to verify the signature

21

Giovanni Vigna, Advanced Topics in Security

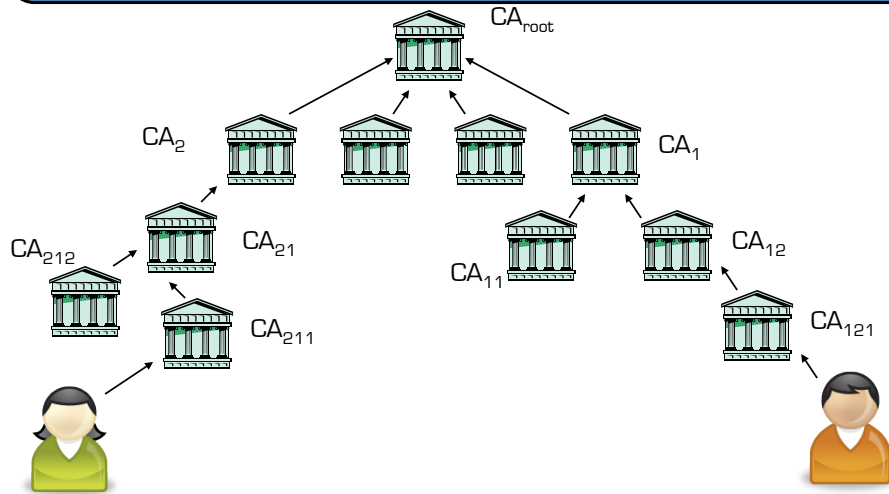
Communication Using Certificates



22

Giovanni Vigna, Advanced Topics in Security

Certifying Certification Authorities (Public Key Infrastructure - PKI)



Conclusions

- Cryptography provides the building blocks for secure communication
- Algorithms
 - Symmetric algorithms
 - Public-key algorithms
 - Hash functions
- Cryptographic protocols use crypto algorithms to provide
 - Confidentiality
 - Integrity
 - Authentication
 - Non repudiation

Weak Crypto

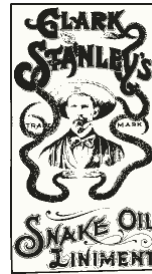
- Researchers have found problems in hash functions
 - MD5 is to be considered insecure for anything that is security-critical
 - It was found to be not collision resistant
 - SHA-1 has problems, too
- Example
 - Alice creates two contracts C1 (favorable to Bob) and C2 (favorable to Alice)
 - Creates variants of C1 and C2 until she finds a collision for C1' and C2'
 - Presents C1' to Bob and gets $H(C1')$ signed by him
 - Presents $S_B(H(C1'))$ to a third party, claiming that Bob signed C2'
- The NIST has launched a competition to select the new SHA-3 hash function
 - Last meeting: August 2010, Santa Barbara

Weak Crypto

- Weak (predictable) seeds in pseudorandom number generators
 - `/dev/random` in Linux/FreeBSD-based systems provides good random numbers based on the "environmental noise" from the OS internals (such as device drivers)
- Choice of a good security random/hash function depend on the bits of (information) entropy
 - Information entropy represent the amount of information/uncertainty carried by a message (e.g., a coin has 1 bit of entropy because the unknown outcome can be represented by 1 bit)
 - Low entropy: predictable
 - High entropy: unpredictable
 - Sometimes a "random" piece of information is not as random as its size might suggest

Bad Crypto

- Snake Oil Crypto
 - <http://www.schneier.com/crypto-gram-9902.html#snakeoil>
- Warning Sign #1: Pseudo-mathematical gobbledygook
- Warning Sign #2: New mathematics
- Warning Sign #3: Proprietary cryptography
- Warning Sign #4: Extreme cluelessness
- Warning Sign #5: Ridiculous key lengths
- Warning Sign #6: One-time pads
- Warning Sign #7: Unsubstantiated claims
- Warning Sign #8: Security proofs
- Warning Sign #9: Cracking contests



Bad Crypto

- From: Meganet <http://www.meganet.com/>:
The base of VME is a Virtual Matrix, a matrix of binary values which is infinity in size in theory and therefore have no redundant value. The data to be encrypted is compared to the data in the Virtual Matrix. Once a match is found, a set of pointers that indicate how to navigate inside the Virtual Matrix is created. That set of pointers (which is worthless unless pointing to the right Virtual Matrix) is then further encrypted in dozens other algorithms in different stages to create an avalanche effect. The result is an encrypted file that even if decrypted is completely meaningless since the decrypted data is not the actual data but rather a set of pointers. Considering that each session of VME has a unique different Virtual Matrix and that the data pattern within the Virtual Matrix is completely random and non-redundant, there is no way to derive the data out of the pointer set

Bad Crypto Implementations

- In May 2008 it was found that the openssl implementation in Debian was broken
 - A code snippet was generating warnings in static analysis tools
 - They removed the snippet!
 - The source of entropy was reduced to the PID of the process (32768 possible values)



Giovanni Vigna, Advanced Topics in Security

29

Bad Crypto Usage

- “How I met your girlfriend”, by Samy Kamkar (BlackHat 2010)
- Identified how PHP session cookies are created

```
session.c
PHPAPI char *php_session_create_id(PS_CREATE_SID_ARGS) /* {{{ */
{
    /* ... code ... */
    gettimeofday(&tv, NULL);
    /* ... code ... */
    sprintf(buf, 0, "%.15s%ld%ld%0.8f",
        remote_addr ? remote_addr : "", /* IP Address: 32 bits */
        tv.tv_sec, /* epoch: 32 bits */
        (long int)tv.tv_usec, /* microseconds: 32 bits */
        php_combined_lcg(TSRMLS_C) * 10); /* rand lcg_value: 64 bits */
    /* ... code ... */
    /* TOTAL: 160 bits */

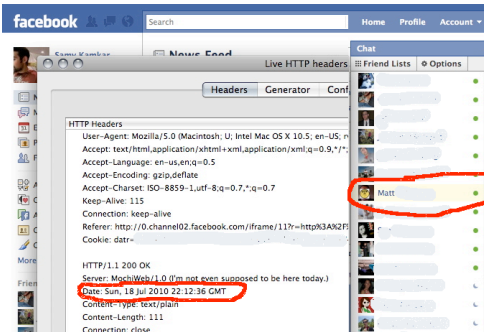
    PHP_SHA1Update(&sha1_context, (unsigned char *) buf, strlen(buf));
    /* ... code ... */
    /* SHA1 string: 160 bits */
}
```

Giovanni Vigna, Advanced Topics in Security

30

Bad Crypto Usage

- IP address (32 bits of entropy)
- Microseconds can only be 0-999,999 (~20 bits of entropy)
- Random value (64 bits of entropy)
- Epoch can be guessed by monitoring when somebody comes online (0 bits of entropy)
- Total: 116 bits

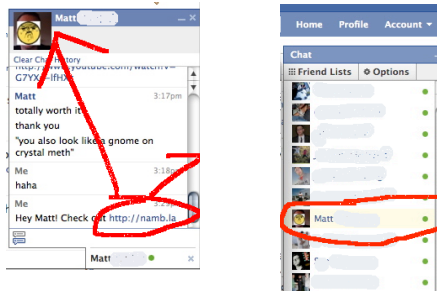


Giovanni Vigna, Advanced Topics in Security

31

Bad Crypto Usage

- The IP address of the user can be stolen by luring the user to connect to a web site under the attacker's control
- Total left: 84 bits



```

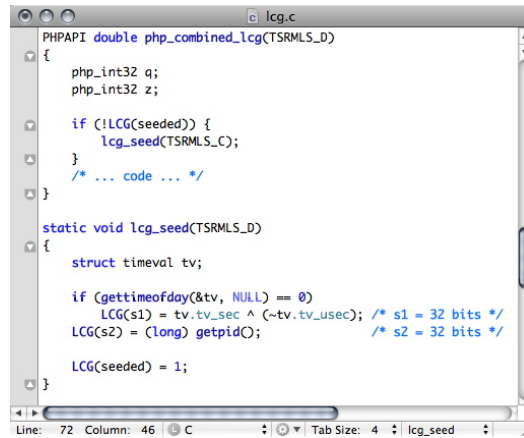
$ cat /var/log/apache2/access.log | tail -f -n 0 access.log
64.134.227.80 - - [18/Jul/2010:22:35:18 +0000] "GET / HTTP/1.1" 200 146 "-"
Mac OS X 10.5; en-US; rv:1.9.2.6) Gecko/20100625 Firefox/3.6.6"
64.134.227.80 - - [18/Jul/2010:22:35:18 +0000] "GET /favicon.ico HTTP/1.1" 404
osh; U; Intel Mac OS X 10.5; en-US; rv:1.9.2.6) Gecko/20100625 Firefox/3.6.6"
    
```

Giovanni Vigna, Advanced Topics in Security

32

Bad Crypto Usage

- PHP's seeding procedures are broken:
 - It uses a composition of time and the PID of the process



```
PHPAPI double php_combined_lcg(TSRMLS_D)
{
    php_int32 q;
    php_int32 z;

    if (!LCG(seeded)) {
        lcg_seed(TSRMLS_C);
    }
    /* ... code ... */
}

static void lcg_seed(TSRMLS_D)
{
    struct timeval tv;

    if (gettimeofday(&tv, NULL) == 0)
        LCG(s1) = tv.tv_sec ^ (~tv.tv_usec); /* s1 = 32 bits */
        LCG(s2) = (long) getpid(); /* s2 = 32 bits */

    LCG(seeded) = 1;
}
```

Bad Crypto Usage

- Only the lower 20 bits matter (for some long period of time)

$$LCG(s1) = tv.tv_sec \wedge (\sim tv.tv_usec)$$

$$LCG(s1) = epoch \wedge (\sim [20 \text{ random bits}])$$

$$\begin{aligned} -0 &= 11111111111111111111111111111111 \\ -1,000,000 &= 11111111111100001011110110111111 \text{ (same 12 bits)} \end{aligned}$$

$$\begin{aligned} epoch &= 1279493871 \\ epoch &= 01001100010000111000011011101111 \text{ (static / unknown)} \\ epoch^{\wedge} &= 01001100010000000000000000000000 \\ epochv &= 0100110001001111111111111111111111 \end{aligned}$$

$$\begin{aligned} epoch^{\wedge} &= \text{Thu Jul 15 23:45:20 2010} \\ epochv &= \text{Wed Jul 28 03:01:35 2010} \\ epoch \text{ diff} &= 12+ \text{ days} \end{aligned}$$

Bad Crypto Usage

- The process ID can only be up to 32,768 and therefore only 15 of the 32 bits matter
- Result: the PRNG seed is only 35 bits
- If one learns the PID (e.g., by executing PHP code on the server) it goes down to 20 bits
- Total bits of entropy left: 40

Bad Crypto Usage

- If the attacker can execute `log_value()`, it is possible to determine the seed (pre-computing the sequence)
- Total bits of entropy left: 20
- Twenty bits correspond to 1,048,576 cookies
 - Can be brute-forced in a reasonable amount of time

Secure Messaging and OpenPGP

- Pretty Good Privacy (PGP) is an application for secure messaging originally developed by Philip Zimmerman
- OpenPGP (RFC 2440) standardized the message-exchange packet formats used to provide encryption, decryption, signing, and key management functions
- Open software applications, such as GnuPG, use the standard to achieve interoperability
- OpenPGP provides data integrity services for messages and data files by using these core technologies
 - Digital signatures
 - Encryption
 - Compression and radix-64 conversion (ASCII armor)

Users and Keys

- Each user owns one or more private key/public key pairs
- Optionally, the private keys can be encrypted using a password
- Private and public keys are stored in a *keyring*
- To generate a key pair (GnuPG)
% **gpg --gen-key**
- To create an ASCII version of a public key that can be put on a web page or sent to a database (GnuPG)
% **gpg --export --armor**
- The key is self-signed, that is, the public key is signed using the corresponding secret key

A Public Key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.2.1 (GNU/Linux)

mQGIBD7mK/MRBAC0LmdSYMw/V1yBa2uFY16gmTt1ydk35CFM+LaiymvYJ9tcVNX5
+id4YkKeQzRigg7RCTbfgUjw23zXapUpIZgIK3RvXbMVvsxdqlotQkZyRf/7VZSml
EVnlflNaqy7O3tjVbNCbEE10GAGp5UwzYfTcix44mYYNdGhcvkbgq5GXfWCgzpsq
CuNqq0+CyFGhI9S1R+aIya8D/0iNMYyU3M5YDhNyRs5Gtjebhc2eGEXCR71QjRww
qNyWTlr43nsF1A8ckPdeqjdlvrSp/F9eFAIcdULZmyDYlMFUSYO2X17xZBNqgDAu
jnrduuuu413owqVILFIhgs8EwNNA0bVSxwgfCQsGWuJr5cNXTggZVKdsqfRyR5
reh4A/9wJcXXOulUNpEbUmrDEMVeNvedEdIh4nQdyxilpcggAhc2WxiFu1NOSYsL
daKLDB4kiP3xfdK7Cq+scvvx6NdKsSDWYibQ0uMfvYDzX3LMKpBBvKxDyD0pfJjg
DLSe9cn7Tp03b7kTKfRC0Fhyd7ECr7cqw6vsnf3cBjaN1YSbQ1R21vdmFubmkg
VmlnbmEgPHZpZ25hQGNzLnVjc2IuZWR1PohZBBMRAgAZBQI+5ivzBasHawIDFQID
AxYCAQIEAQIXgAAKRCrju5XfPpVYj02AKDDsYft/N00g+v2511Laxpzv9Dp0ACD
FTTMe4I3mCI6zW38v62OECHPR614zQQ+5iv0EAMAn8y7knVSp1raHNdGzWUC+i4h
4TB15DBxzCBS591K28ef6+olrBnSJBdP5370vGxvUNczRjwFXnRcto9et3enraM
mF0SxD+qadyFdukeFJr5cqdFMWTwwjh9efRsS3JnAAMFAv9KTUYSUjTFRqjw8+H
4npDq6Vvy7cXo/3hHLGXhO2DGu9x1RbglFyeSoMbgq2ALjLhJJs1I1NemhySN8ZfO
DdC9WITYKDasCcIj4b5dcgkxMg0XSwi7FRToVdyww2jIWKuIRgQVEQIABgUCPuYr
9AAKRCrju5XfPpVYt3tAJ9LT1PibtwUEyZPNyrDFWsyvSwkDgCgqOOGkfYzZgQE
JOLKi4c/eFOM5cs=
=/Ufb
-----END PGP PUBLIC KEY BLOCK-----
```

Encryption of a Message

- The sender creates a message
- The sender generates a random number to be used as a session key for this message only
- The session key is encrypted using each recipient's public key. These "encrypted session keys" start the message
- The sender encrypts the message using the session key, which forms the remainder of the message
 - Note that the message is also usually compressed
- The recipient decrypts the session key using the recipient's private key
- The recipient decrypts the message using the session key
 - If the message was compressed, it will be decompressed

Signing of a Message

- The sender creates a message
- The sender generates a hash code of the message
- The sender generates a signature from the hash code, that is, the sender encrypts the hash code with the his/her private key
- The binary signature is attached to the message
- The recipient decrypts the message signature with the sender's public key and keeps a copy of the hash code
- The recipient generates a new hash code for the received message and compares it with the one contained in the message's signature

Signed Message

```
% gpg --clearsign msg.txt
% cat msg.txt.asc

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

ciao
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org

iD8DBQE8p4ZzwsUmGjc7nDERAjYfAKCglVFIisuXJrh1JXcEahesYTngdSwCglbGi
yZHp0X0roCsv0rM8MxCAF0=
=rfdZ
-----END PGP SIGNATURE-----
```

Key Management

- The model introduced by PGP is different from the well-known PKI-based model
- In order to use somebody's public key one should sign it to express that he/she is sure about the identity/key association
- It is also possible to *trust* somebody else to sign public keys
 - Alice trusts Bob as a signer
 - Alice receives Eve's public key signed by Bob
 - Alice accepts Eve's key as valid without further checks
- This procedure creates a *web of trust*
- Different levels of trust can be used
 - "Accept keys that are signed by three people that I trust 70%"

Revoking a Key

- What if the private key is compromised, lost, or the password that protects the key is forgotten?
- At creation time it is possible to create a revocation message
- When distributed to recipients the revocation message invalidates the corresponding public key
- Recipients will not be allowed to use that public key (contained in their keyrings) to check signatures

Common Operations

- Encrypt a message with a secret key
- Encrypt a message with the public key of a single recipient
- Encrypt a message with the public key of multiple recipients
- Sign a message in clear form
- Sign and encrypt a message for a single user

Alice and Bob (and Eve)

