Chapter 6. Public Key Cryptography©

Notice: this chapter needs substantial more work. To add: intro; key agreement; RSA; bit of number theory; finite-use (hash-based) signatures and their use; figures; …

6.1. Some number theory

6.2. Public Key Distribution

Describe problem and DH protocol. Define the DH distinguishability assumption (cannot distinguish btw. (g^x,g^y,g^z) and (g^x,g^y,g^xy)). Define disc-log assumption and show it is weaker than DH.

6.3. Public Key Cryptosystems

By adopting the Kerckhoffs' principle, the design of shared key cryptosystems (ciphers) is available to the adversary, hence public knowledge, which facilitates standardization and mass production. However, shared key cryptosystems still require the use of a secret key. Clearly, decryption requires a secret key, as we do not want an adversary to be able to decrypt the ciphertext. Diffie and Hellman, in a seminal paper [DH76], noted that if encryption and decryption used different keys, then secrecy depends only on keeping the decryption key hidden. Furthermore, there are several important advantages in making the encryption key public rather than secret, e.g.:

- We can keep a public directory of encryption keys, allowing the sender to encrypt without any prior communication with the recipient.
- Either the recipient, or any trusted server, can send the public key of the recipient to the sender. This allows the sender and recipient to be in remote locations connected only via an insecure channel (e.g. the Internet), a common situation in e-commerce. (Diffie and Hellman presented in [DH76] a key exchange protocol that can establish a shared key over insecure channel; we describe it later.)
- When communication is among many peers, sending one set of public keys to everybody is much easier than sharing secret keys among all pairs.

Public key cryptosystems use two, related but different, keys – public key for encryption, and private key for decryption. A metaphor for public key cryptosystems is of a private mailbox; everybody knows the number of the box (public key), but only the owner has the key to open the box (private key).

Definition 1 A *public key cryptosystem* is defined by three efficient algorithms:

1. *KeyGenerate*, whose inputs are a random string and the required key length, and whose output is a pair <*Pub*,*Priv*> of a public key *Pub* and a corresponding private key *Priv*.
2. *Encrypt*, whose inputs are a plaintext block *p* and the public key *Pub*, and whose output is a corresponding ciphertext block $c=Encrypt_{Pub}(p)$.
3. *Decrypt*, whose inputs are a ciphertext block *c* and the private key *Priv*, and whose output is the (decrypted) plaintext block, i.e.
$$p=Decrypt_{Priv}(c)= Decrypt_{Priv}(Encrypt_{Pub}(p)).$$

A public key cryptosystem is *deterministic* if the Encrypt algorithm is deterministic, and *probabilistic* if the Encrypt algorithm is probabilistic (with random input, in addition to the plaintext and public key).

As for block ciphers, a deterministic public key cryptosystem can only be secure for specific probability distributions of the plaintext, e.g. for uniformly distributed plaintext. Therefore, a more appropriate term for deterministic public key cryptosystems, used by many researchers, is *pseudorandom permutations with trapdoor*.

6.3.1. Practical Public Key Cryptosystems – RSA and others

Design of public key cryptosystems appear much more difficult than design of shared key cryptosystems (ciphers). A public key cryptosystem involves asymmetry in the operation of the keys (indeed, it is sometimes referred to as an asymmetric cryptosystem). The asymmetry requires the private key to invert the encryption operation, performed via the public key, and not invertible without the private key. Namely, the private key operates as a `secret trapdoor`.

Only the concept of public key cryptosystems appeared in [DH76]. Later, Rivest, Shamir and Adleman presented the first public key cryptosystem, in [RSA78], named RSA (after their initials). RSA is based on number theory, and specifically on the supposed computational hardness of the factoring problem. RSA is still the most important and most widely used public key cryptosystem. There were several other proposals for public key cryptosystems; some were broken and others, like RSA, withstood cryptoanalysis efforts so far.

Discuss effective key length

6.3.1.1. Choosing public exponent

6.3.1.2. Commutative property

6.3.1.3. Elliptic Curve Cryptosystems

6.3.2. Efficiency considerations and hybrid cryptosystems

All of the proposed public key cryptosystems are substantially more computationally intensive than private key cryptosystems. Therefore, a standard technique is to encrypt with the public key only a randomly chosen secret shared, and use this shared key to

encrypt the plaintext. This is usually much more efficient than encrypting the entire plaintext using the public key. The ciphertext of this technique is the concatenation of the ciphertexts produced by the shared key and the public key encryptions, namely *SKCiphertext* and *CipherKey* correspondingly (see Figure 6.1).

We call this technique *hybrid encryption*, and the resulting system a *hybrid cryptosystem.* The decryption process involves two steps. The first step is decryption of the encrypted shared key, $key' = Decrypt_{PRIV}(CipherKey)$, where $key'=key$ if $CipherKey=Encrypt_{PUB}(key)$ as designed. The second step is decryption of the encryption of the plaintext using the decrypted key *key'*, namely $Plaintext'=Decrypt_{key'}(SKCiphertext)$. If the key was decrypted correctly ($key'=key$) and $SKCiphertext=Encrypt_{key}(Plaintext)$ as designed, then the decryption is correct, i.e. $Plaintext'= Plaintext$.

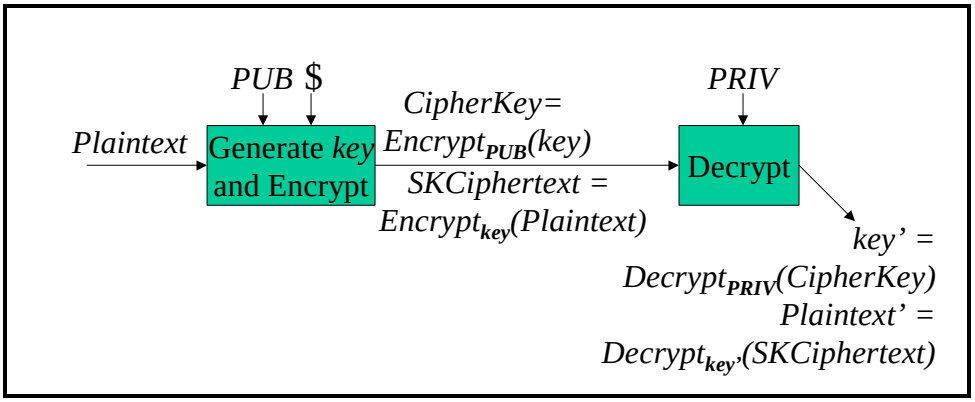

Figure 6.1: Hybrid encryption

6.3.3. Key Length for Practical Public Key Cryptosystems

The core of RSA, and most other public key cryptosystems, is a number theoretic problem. For RSA, this problem is the factoring of very large numbers. Part of the computational cost of RSA is due to its use of long key (and hence exponentiation of very large numbers). This is required as there are efficient factorization algorithms for rather long numbers, although the best-known asymptotic solutions are still exponential. As a result, 1024 bits is a typical key length used today for RSA secret key, and only 128 bits for shared key cryptosystems. In RSA (and many other public key cryptosystems), encryption blocks are of the same length as the key (e.g. 1024 bits). Therefore, with RSA, and many other public key cryptosystems, both keys and encryption blocks are of substantial size (e.g. 1024 bits, compared with say 128 bits for shared key cryptosystems). This may cause significant overhead for some applications, e.g. storage of the secret key on a smart-card chip with very limited memory. Some of the alternative public key cryptosystems, e.g. based on elliptic curves, try to address this problem, by using other computationally hard problems that allow shorter keys.

6.3.4. Probabilistic public key cryptosystems

6.3.5. TBD: Public key cryptosystem encoding

PKCS 1 (& variants)

6.4. Public key digital signatures

Message Authentication Code (MAC) can ensure a recipient of a message that the message received was previously sent by a specific sender, and was not modified en-route in any way. However, this does not provide non-repudiation. Namely, the recipient knows that the message was sent by somebody with access to the secret key, which presumably is held only by the sender and the recipient, and therefore can usually infer that it was sent by the sender. But, the recipient cannot prove this to a third party (e.g. judge).

Non-repudiation is critical for many commercial applications, as it provides a way to resolve disputes between the parties without having to decide who is providing an accurate account. In classical, non-cryptographic commerce, non-repudiation is achieved, usually, by using handwritten signatures in pen over paper; we call these *physical signatures*. Physical signatures provide a way to commit a person to his statement or agreement. Notice that it is not always trivial to detect modifications in the document performed after signing it, as well as forged physical signatures. Indeed, such issues are sometimes a cause for debate among the parties and their resolution may involve length and expensive legal process and experts in physical signature and counterfeiting. Furthermore, the resolution techniques can only be applied to the originally signed paper document. Copies and facsimiles of the document do not, in truth, provide non-repudiation. In fact, it is possible to create modified copies, so that even an expert will not be able to identify that they are not original, when comparing to unmodified copies.

Public key digital signatures, or simply digital signatures, also allow a party (not necessarily a person) to commit to a document. However, digital signatures allow complete automation of the validation and signature processes. Furthermore, digital signatures, and the signed document, are both represented as a string of bytes (octets). Therefore, they can be easily copied, transmitted over a network and archived. These are huge advantages, especially for performing commerce over the Internet.

Public key digital signatures, like public key cryptosystems, involve a pair of keys for each party: a public key and a private, secret key. In a digital signature scheme, the secret key is used to *sign* documents, and the public key is used to *validate* the signature. Figure 6.2 illustrates digital signing of document *m* by Alice, and subsequent validation by Bob. If Bob receives the message *m* and the *signature tag* $Sign(S_A,m)$ exactly as sent, i.e. it receives *m* and $s=Sign(S_A,m)$, then $Valid(m,s)$ returns *true*. Otherwise, $Valid(m,s)$ returns *false.*

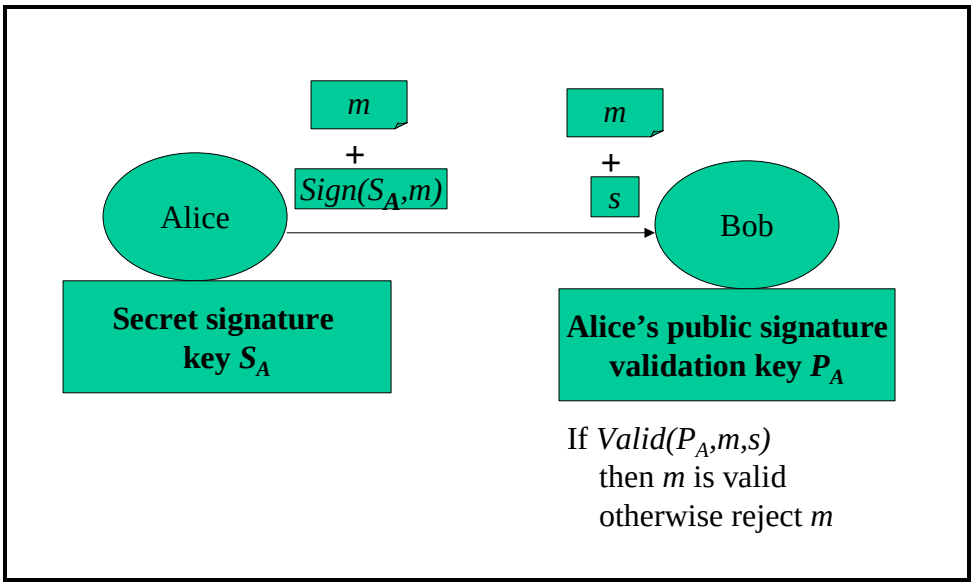

Figure 6.2: Public Key Digital Signature

In Figure 6.3 we illustrate adversary Eve, trying to create a message and a signature tag appearing to be signed by Alice. This appears quite similar to the scenario with MAC functions, as in **Error! Reference source not found.**. Again, the adversary is unable to find a tag that validates correctly for any different message. Notice that with digital signatures, the adversary normally knows Alice's public validation key, P_A. However, this should not help the adversary in forging a signature.

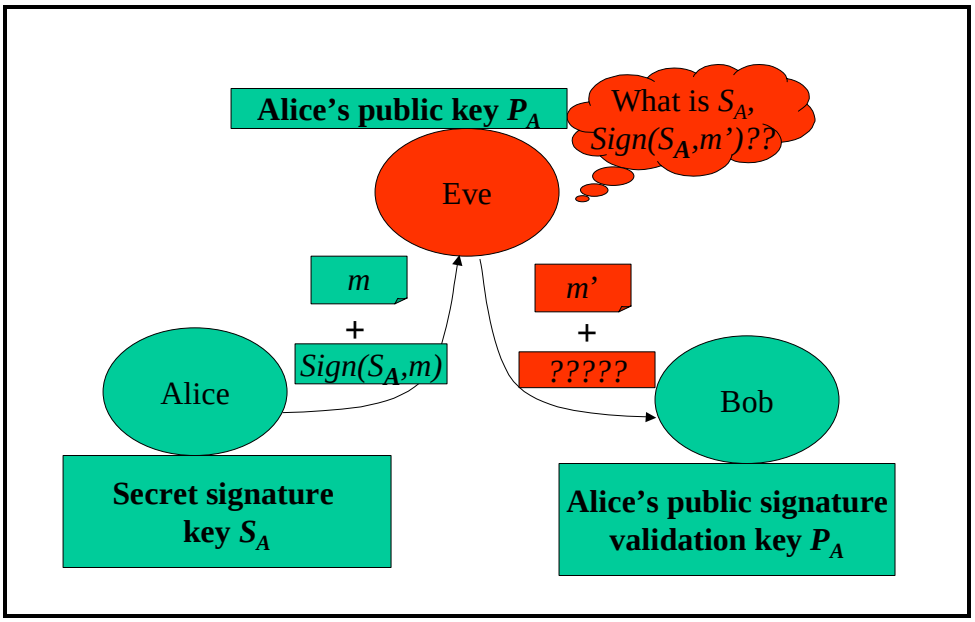

Figure 6.3: Adversary Eve trying to forge a signature

Physical signatures are not really a good metaphor for public key digital signatures, as there is no direct linkage between them and the signed document, and also there is no real `public key` involved. A seal may be a better metaphor; the assumption is that a sealed

document cannot be modified (without breaking the seal). Furthermore, while everybody may recognize the imprint (appearance) of the seal, it is assumed to be infeasible to generate another seal which with this imprint. This assumption, admittedly, is not realistic when it comes to seals, which with the right know-how can in fact be re-create from an imprint.

We now define a digital signature system. As for Definition 4.2 of public key cryptosystems, we again consider three algorithms – sign, validate, and key generation.

Definition 2 A *public key digital signature system* is defined by three efficient algorithms:

4. *KeyGenerate*, whose inputs are a random string and the required key length, and whose output is a pair <*Pub*,*Priv*> of a public key *Pub* and a corresponding private key *Priv*.
5. *Sign*, whose inputs are a message *m* and the private key *Priv*, and whose output is the signature tag *s=Sign(Priv,m)*
6. *Validate*, whose inputs are a message *m*, a signature tag *s*, and the public key *Pub*, and whose output is *true* if *s* is a possible output of *Sign(Priv,m)* and *false* otherwise.

While we only specified *KeyGenerate* as having randomized input, the other algorithms, and in particular *Sign*, can also be randomized. To allow for randomized *Sign* algorithm, the validation was defined as *true* if *s* is any possible output of *Sign(Priv,m)* – if it is randomized, there may be alternate outputs to the same signed message *m.* In fact, the *Sign* algorithm is often randomized in practice as well as in theoretical constructions.

We consider the following classes of attack models:

- *Key-Only Attack*: In this attack the adversary knows only the public key of the signer and therefore only has the capability of checking the validity of signatures of messages given to him.
- *Known Signature Attack:* The adversary knows the public key of the signer and has seen message/signature pairs chosen and produced by the legal signer. This attack is usually possible in practice, and therefore any signature scheme must be secure against it.
- *Chosen Message Attack*: The adversary is allowed to ask the signer to sign a number of messages of the adversary's choice. The choice of these messages may depend on previously obtained signatures. This attack may be impractical in most cases, but following the prudence principle, it is preferable to use a signature scheme secure against it. For example, one may think of a notary public who signs documents on demand.

We also consider different definitions for a successful attack:

- Existential Forgery: The adversary succeeds in forging the signature of one message, not necessarily of his choice.

- Selective Forgery: The adversary succeeds in forging the signature of some message of his choice.
- Universal Forgery: The adversary, although unable to find the secret key of the forger, is able to forge the signature of any message.
- Total Break: The adversary can compute the signer's secret key.

6.4.1. Hash and Sign

Public key algorithms, and in particular digital signature, typically requires substantially more computational resources than shared key or hashing functions, with comparable security. As a result, it is desirable to replace at least some of the public key operations, with shared key or hashing functions.

One standard technique is to hash messages before signing them. This is almost always done in practice. Let *Sign* be the digital signature algorithm, and *h* be a weak CRHF. Then *Sign(Priv,h(m))* is also a digital signature algorithm.

In some cases, we need to sign two related documents, e.g., a purchase order (to the merchant) which also include the payment details (to the bank). For example, say that the signed information contains common content m_c, as well as specific content m_1 and m_2, which only some recipient or applications need to know. Then we could use the signature

$s=Sign(m_c, h(m_1), h(m_2))$

Recipients that need only m_c receive $\{s, m_c, h(m_1), h(m_2)\}$.
Recipients that need only m_c and m_2 (or m_1) receive $\{s, m_c, m_2, h(m_1)\}$ (respectively $\{s, m_c, m_1, h(m_2)\}$.

Recipients that need both m_2 and m_1 receive $\{s, m_c, m_2, m_1\}$.

Another way in which hashing can extend signatures is by hash chains. Suppose Alice needs to send Bob multiple signed documents, e.g. each of them a payment for one cent. Instead of signing each payment individually, Alice signs one statement containing a hash chain:

$$Sign(PRIV_{Alice}, \text{"I pay Bob one cent for each pre-image"}, h^{100}(x))$$

Later, to pay the first cent, Alice sends Bob the value of $h^{99}(x)$; and so on until sending x (to pay the maximal amount, which is one dollar in this case).

Notice that in all of these applications, the hash function need only be weakly CRHF. This is since Bob is given the hash and its pre-image; Alice cannot change her mind and present a different pre-image later.

6.5. Exercises

1. A proposal is made to perform hybrid authentication, in the same manner as hybrid encryption, but authenticating the message using MAC instead of encrypting it. Namely the sender selects *key* randomly and sends $CipherKey=Encrypt_{\boldsymbol{PUB}}(key)$ as in

Figure 6.1, but appends to it *msg*, $MAC_{key}(msg)$ for authenticating message *msg*.
Criticize: Is this solution secure? Is there a better way to authenticate a long message with a single public key operation?

2. Consider RSA signatures where messages are hashed and then raised by the private key, i.e. $Sign_d(m)=(h(m))^d \bmod n$. Show a weakness with these signatures, when *h()* is *not Multiplicative*-resistant hash function, where *Multiplicative(a,b,c)=True* if and only if $ab=c$.