Key Management

Sachin Tripathi

IIT(ISM), Dhanbad

Outline

- Identify some fundamentals principles of key management
- Discuss some key establishment mechanisms

Security Services

- The basic security services are achieved in the form of:
  - Confidentiality (with encryption algorithms)
  - Integrity (with MACs or digital signatures)
  - Message authentication (with MACs or digital signatures)
  - Non-repudiation (with digital signatures)

Key Management

- Key management is crucial to the security of any cryptosystem.
- Without secure procedures for handling of cryptographic keys throughout their lifecycle, the benefits of the use of strong cryptographic primitives are potentially lost.
- If key management is not performed correctly then there is no point in using cryptography at all.

Key Lifecycle

- Key Generation: concerns the creation of keys.
- Key Establishment: is the process of making sure that keys reach the end points where they will be used.
- Key Storage: deals with the safekeeping of keys. It may also be important to conduct key backup so that keys can be recovered in the event of loss of a key.

Key lengths

- In general, longer keys are better from a security perspective.
- A cryptographic computation normally takes more time if the key is longer.
- In addition, longer keys involve greater storage and distribution overheads.
- Longer keys are less efficient in several important respects.
- Thus key length tends to be based on an efficiency-security trade-off.

Key lifetimes

- Every key has certain lifespan for a particular application.
- There are many reasons why cryptographic keys have finite lifetimes.
  - Mitigation against key compromise:
  - Mitigation against key management failures
  - Mitigation against future attacks
  - Enforcement of management cycles
  - Flexibility
  - Limitation of key exposure

Key Establishment

- How keys can be established using symmetric cryptosystems?
- How keys can be established using public-key cryptosystems?
- Why public-key techniques still have shortcomings for key distribution?
- What certificates are and how they are used?
- The role that public-key infrastructures play

Continued..

- All cryptographic mechanisms that we have introduced so far assume that keys are properly distributed between the parties involved, e.g., between Alice and Bob.
- The task of key establishment is in practice one of the most important and often also most difficult parts of a security system.
- We already learned some ways of distributing keys, in particular Diffie–Hellman key exchange.
- We will learn many more methods for establishing keys between remote parties.

Contd…

- Key establishment deals with establishing a shared secret between two or more parties.
- Methods for this can be classified into key transport and key agreement methods.
- A key transport protocol is a technique where one party securely transfers a secret value to others.
- In a key agreement protocol two (or more) parties derive the shared secret where all parties contribute to the secret.

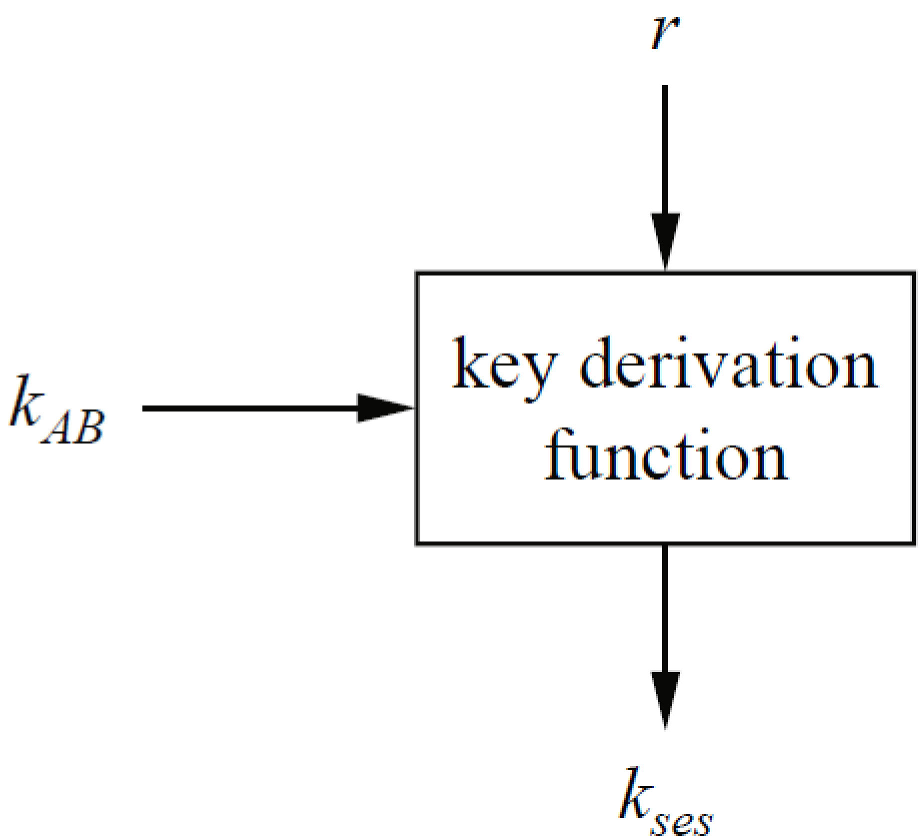
Key Freshness

- It is desirable to use cryptographic keys which are only valid for a limited time, such keys are called *session keys.*
- A major advantage is that there is less damage if the key is exposed.

Implementation issue

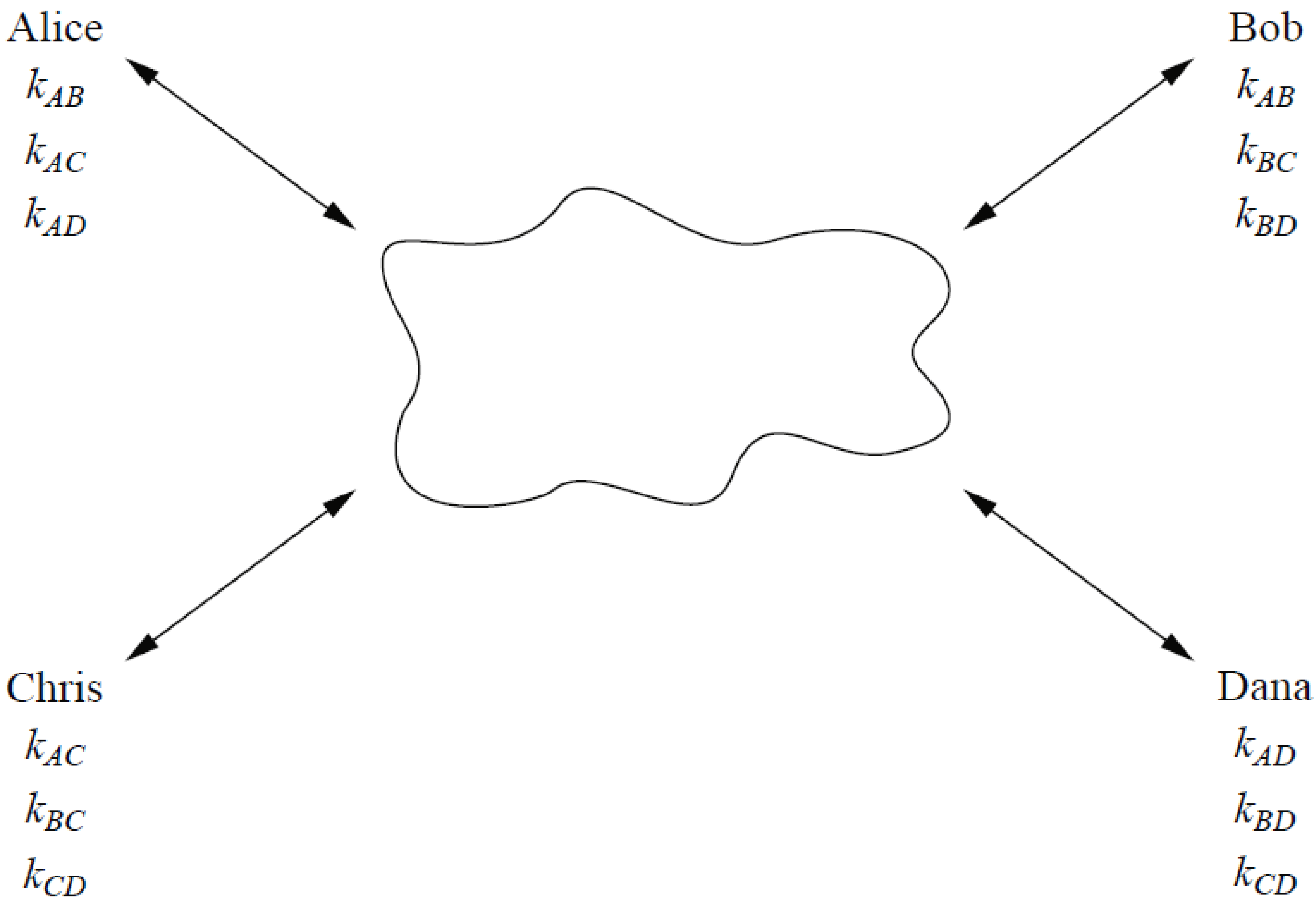
- There are always certain costs associated with key establishment, typically with respect to additional communication connections and computations.
- The latter holds especially in the case of public-key algorithms which are very computationally intensive.

Key Derivation



Key Derivation with Nonces

Alice		**Bob**
		generate nonce r
	$\xleftarrow{\quad r \quad}$	
derive key		derive key
$k_{ses} = e_{k_{AB}}(r)$		$k_{ses} = e_{k_{AB}}(r)$



Keys in a network with $n = 4$ users

Contd…

- Each user must store $n-1$ *keys.*
- There is a total of $n(n-1) \approx n^2$ *keys in the network*.
- A total of $n(n-1)/2 = {}^nC_2$ symmetric key pairs are in the network.
- If a new user joins the network, a secure channel must be established with every other user in order to upload new keys.

Key Establishment with KDC

Basic Key Establishment Using a Key Distribution Center

Alice		KDC		Bob
KEK: k_A		KEK: k_A, k_B		KEK: k_B
	$\xrightarrow{\text{RQST}(ID_A, ID_B)}$			
		generate random k_{ses}		
		$y_A = e_{k_A}(k_{ses})$		
		$y_B = e_{k_B}(k_{ses})$		
	$\xleftarrow{y_A}$		$\xrightarrow{y_B}$	
$k_{ses} = e_{k_A}^{-1}(y_A)$				$k_{ses} = e_{k_B}^{-1}(y_B)$
$y = e_{k_{ses}}(x)$	$\xrightarrow{y}$			$x = e_{k_{ses}}^{-1}(y)$

Alternative

Key Establishment Using a Key Distribution Center

Alice		**KDC**		**Bob**
KEK: k_A		KEK: k_A, k_B		KEK: k_B
	$\xrightarrow{\text{RQST}(ID_A, ID_B)}$			
		generate random k_{ses} $y_A = e_{k_A}(k_{ses})$ $y_B = e_{k_B}(k_{ses})$		
	$\xleftarrow{y_A, y_B}$			
$k_{ses} = e_{k_A}^{-1}(y_A)$ $y = e_{k_{ses}}(x)$				
	$\xrightarrow{y, y_B}$			
				$k_{ses} = e_{k_B}^{-1}(y_B)$ $x = e_{k_{ses}}^{-1}(y)$

Replay Attack

- This attack makes use of the fact that neither Alice nor Bob know whether the encrypted session key they receive is actually a new one.
- If an old session key is reused, key freshness is violated.

Key Confirmation Attack

Alice
KEK: k_A

Oscar
KEK: k_O

KDC
KEK: k_A, k_B, k_O

Bob
KEK: k_B

$\xrightarrow{\text{RQST}(ID_A, ID_B)}$

↯ substitute

$\xrightarrow{\text{RQST}(ID_A, ID_O)}$

random k_{ses}
$y_A = e_{k_A}(k_{ses})$
$y_O = e_{k_O}(k_{ses})$

$\xleftarrow{y_A, y_O}$

$k_{ses} = e_{k_A}^{-1}(y_A)$
$y = e_{k_{ses}}(x)$

$\xrightarrow{y, y_O}$

↯ intercept
$k_{ses} = e_{k_O}^{-1}(y_O)$
$x = e_{k_{ses}}^{-1}(y)$

Kerberos

Key Establishment Using a Simplified Version of Kerberos

Alice		**KDC**		**Bob**
KEK: k_A		KEK: k_A, k_B		KEK: k_B
generate nonce r_A				
	$\xrightarrow{\text{RQST}(ID_A, ID_B, r_A)}$			
		generate random k_{ses} generate lifetime T $y_A = e_{k_A}(k_{ses}, r_A, T, ID_B)$ $y_B = e_{k_B}(k_{ses}, ID_A, T)$		
	$\xleftarrow{y_A, y_B}$			
$k_{ses}, r'_A, T, ID_B = e_{k_A}^{-1}(y_A)$ verify $r'_A = r_A$ verify ID_B verify lifetime T generate time stamp T_S $y_{AB} = e_{k_{ses}}(ID_A, T_S)$				
	$\xrightarrow{y_{AB}, y_B}$			
				$k_{ses}, ID_A, T = e_{k_B}^{-1}(y_B)$ $ID_A', T_S = e_{k_{ses}}^{-1}(y_{AB})$ verify $ID_A' = ID_A$ verify lifetime T verify time stamp T_S
$y = e_{k_{ses}}(x)$	$\xrightarrow{y}$			$x = e_{k_{ses}}^{-1}(y)$

Limitations

- Communication requirements
- Secure channel during initialization
- Single point of failure
- No perfect forward secrecy

Key Establishment using Asymmetric Techniques

- Public-key algorithms are especially suited for key establishment protocols since they don't share most of the drawbacks that symmetric key approaches have.
- The public-key primitives are quite slow, and that for this reason actual data encryption is usually done with symmetric primitives like AES or 3DES, after a key has been established using asymmetric techniques.
- At this moment it looks as though public-key schemes solve all key establishment problems.
- It turns out, however, that they all require what is termed an *authenticated channel to distribute the public keys.*

Man-in-the-Middle Attack

- The *man-in-the-middle attack is a serious attack against public-key algorithms.*
- The basic idea of the attack is that the adversary, Eve, replaces the public keys sent out by the participants with his own keys.
- This is possible whenever public keys are not authenticated.

Diffie–Hellman Key Exchange

Alice		**Bob**
choose random $a = k_{pr,A}$		choose random $b = k_{pr,B}$
compute $A = k_{pub,A} \equiv \alpha^a \bmod p$		compute $B = k_{pub,B} \equiv \alpha^b \bmod p$
	$\xrightarrow{A}$	
	$\xleftarrow{B}$	
$k_{AB} \equiv B^a \bmod p$		$k_{AB} \equiv A^b \bmod p$

Man-in-the-Middle Attack Against the DHKE

Alice		**Oscar**		**Bob**
choose $a = k_{pr,A}$				choose $b = k_{pr,B}$
$A = k_{pub,A} \equiv \alpha^a \bmod p$				$B = k_{pub,B} \equiv \alpha^b \bmod p$
	$\xrightarrow{A}$	$\lightning$ substitute $\tilde{A} \equiv \alpha^o$	$\xrightarrow{\tilde{A}}$	
	$\xleftarrow{\tilde{B}}$	$\lightning$ substitute $\tilde{B} \equiv \alpha^o$	$\xleftarrow{B}$	
$k_{AO} \equiv (\tilde{B})^a \bmod p$		$k_{AO} \equiv A^o \bmod p$ $k_{BO} \equiv B^o \bmod p$		$k_{BO} \equiv (\tilde{A})^b \bmod p$

Certificates

- The underlying problem of the man-in-the-middle (MIM) attack is that public keys are not authenticated.
- The MIM attack is not restricted to the DHKE, but is in fact applicable to any asymmetric crypto scheme.
- The attack always proceeds the same way: Eve intercepts the public key that is being sent and replaces it with his own.

Contd…

- The problem of trusted distribution of private keys is central in modern public key cryptography.
- There are several ways to address the problem of key authentication.
- The main mechanism is the use of *certificates.*
- A certificate for a user Alice in its most basic form is the following structure:

$$\mathrm{Cert}_A = [(k_{pub,A}, ID_A), \mathrm{sig}_{k_{pr}}(k_{pub,A}, ID_A)]$$

Contd…

- Certificates require that the receiver has the correct verification key, which is a public key.
- If we were to use Alice's public key for this, we would have the same problem that we are actually trying to solve.
- Instead, the signatures for certificates are provided by a mutually trusted third party.
- This party is called the **Certification Authority (CA)**. It is the task of the CA to generate and issue certificates for all users in the system.
- For certificate generation, the user computes her own asymmetric key pair and merely requests the CA to sign the public key.

Certificate Generation with User-Provided Keys

Alice		**CA**
generate $k_{pr,A}, k_{pub,A}$		
	$\xrightarrow{\text{RQST}(k_{pub,A}, ID_A)}$	
		verify ID_A
		$s_A = \text{sig}_{k_{pr},CA}(k_{pub,A}, ID_A)$
		$\text{Cert}_A = [(k_{pub,A}, ID_A), s_A]$
	$\xleftarrow{\text{Cert}_A}$	

CA's role

- From a security point of view, the first transaction is crucial.
- It must be assured that Alice's message (k_{pub_A}, ID_A) *is sent via an authenticated channel.*
- *Otherwise, Eve* could request a certificate in Alice's name.
- In practice it is often advantageous that the CA not only signs the public keys but also generates the public–private key pairs for each user.
- In this case, a basic protocol looks like this:

Certificate Generation with CA-Generated Keys

Alice		**CA**
request certificate	$\xrightarrow{\text{RQST}(ID_A)}$	
		verify ID_A generate $k_{pr,A}, k_{pub,A}$ $s_A = \text{sig}_{k_{pr},CA}(k_{pub,A}, ID_A)$ $\text{Cert}_A = [(k_{pub,A}, ID_A), s_A]$
	$\xleftarrow{\text{Cert}_A, k_{pr,A}}$	

Diffie–Hellman Key Exchange with Certificates

Alice		**Bob**
$a = k_{pr,A}$		$b = k_{pr,B}$
$A = k_{pub,A} \equiv \alpha^a \bmod p$		$B = k_{pub,B} \equiv \alpha^B \bmod p$
$\mathrm{Cert}_A = [(A, ID_A), s_A]$		$\mathrm{Cert}_B = [(B, ID_B), s_B]$
	$\xrightarrow{\mathrm{Cert}_A}$	
	$\xleftarrow{\mathrm{Cert}_B}$	
verify certificate:		verify certificate:
$\mathrm{ver}_{k_{pub,CA}}(\mathrm{Cert}_B)$		$\mathrm{ver}_{k_{pub,CA}}(\mathrm{Cert}_A)$
compute session key:		compute session key:
$k_{AB} \equiv B^a \bmod p$		$k_{AB} \equiv A^b \bmod p$

- One very crucial point here is the verification of the certificates. Without verification, the signatures within the certificates would be of no use.
- Verification requires the public key of the CA. This key must be transmitted via an authenticated channel, otherwise Eve could perform MIM attacks again.
- We need the authenticated channel only once, at set-up time.
- We saw in the earlier example of DHKE without certificates, that Alice and Bob have to trust each other's public keys directly.
- With the introduction of certificates, they only have to trust the CA's public key k_{pub_CA}.
- If the CA signs other public keys, Alice and Bob know that they can also trust those.
- This is called a chain of trust.

Public-Key Infrastructures (PKI) and CAs

- In order for public key cryptography to be useful in commercial applications, it is necessary to have an infrastructure that keeps that track of public keys.
- A Public-Key Infrastructures (PKI) is a framework consisting of policies defining the rules under which the cryptographic systems operate and procedures for generating and publishing keys and certificates.

X.509 Certificates

Serial Number
Certificate Algorithm: - Algorithm - Parameters
Issuer
Period of Validity: - Not Before Date - Not After Date
Subject
Subject's Public Key: - Algorithm - Parameters - Public Key
Signature