Cryptography and Network Security

Spring 2006
http://www.abo.fi/~ipetre/crypto/

Lecture 8: Protocols for public-key management

Ion Petre
Academy of Finland and
Department of IT, Åbo Akademi University
Key management – two problems

- Distribution of public keys (for public-key cryptography)
- Distribution of secret keys (for classical cryptography)
  - This has more to do with authentication of the users
One problem we went over too quickly: if Alice and Bob do not know each other, how do they get each other’s public key to communicate with each other?

Solution 1: append your public key (e.g., for PGP) to the end of your email
- Attack: emails can be forged – Eve sends an email to Bob pretending she is Alice and handing him a public key, supposedly Alice’s; she will be able to communicate with Bob pretending she is Alice

Solution 2: post it on your website
- Attack: Eve breaks into the DNS server and sends Alice a fake webpage purportedly of Bob’s
- Alice encrypts the message using that public key and Eve will be able to read it; Eve may even modify the message and forwards it to Bob using his public key
First solution: Public-key Authority

- Bottom-line idea here: get the public key of the other user from a trusted central authority
- This scenario assumes the existence of a public authority (whoever that may be) that maintains a dynamic directory of public keys of all users
  - The public authority has its own (private key, public key) that it is using to communicate to users
- Whenever Alice wants to communicate with Bob she will go through the following protocol:
  1. Alice sends a **timestamped** message to the central authority with a request for Bob’s public key (the time stamp is to mark the moment of the request)
  2. The authority sends back a message encrypted with its private key (for authentication) – message contains Bob’s public key and the original message of Alice – this way Alice knows this is not a reply to an old request;
  3. Alice starts the communication to Bob by sending him an encrypted message containing her name and a random number (to identify uniquely this transaction)
  4. Bob gets Alice’s public key in the same way (step 1)
  5. Bob gets Alice’s public key in the same way (step 2)
  6. Bob replies to Alice by sending an encrypted message with Alice’s random number plus another random number (to identify uniquely the transaction)
  7. Alice replies once more encrypting Bob’s random number
- Steps 6, 7 are desirable so that Alice and Bob authenticate each other – indeed they are the only ones who could read each other’s random number
First solution: Public-key Authority

1. Request $\|$ $\text{Time}_1$
2. $E_{KR_{auth}}[KU_b \;\|\; \text{Request} \;\|\; \text{Time}_1]$
3. $E_{KU_b}[\text{ID}_A \;\|\; N_1]$
4. Request $\;\|\; \text{Time}_2$
5. $E_{KR_{auth}}[KU_a \;\|\; \text{Request} \;\|\; \text{Time}_2]$
6. $E_{KU_a}[N_1 \;\|\; N_2]$
7. $E_{KU_b}[N_2]$
Drawbacks to the first solution

- For any communication between any two users, the central authority must be consulted by both users to get the newest public keys
- The central authority must be online 24 hours/day
- If the central authority goes offline, all secure communications halt
- This clearly leads to an undesirable bottleneck
Second solution: Certificate Authority

- **Idea: have a trusted authority to certify one's own public key**
  - Whenever Alice wants to start secure communication with Bob, she sends him her public key certified by the central authority (encrypted with its private key) – Bob will know that it is indeed Alice because he will see her name in the certificate (he can decrypt it using the authority's public certificate).
  - To get her own certificate, she must visit the authority (with her passport) or otherwise use some type of e-security – after that, she may place it on the web because it is unforgeable.
  - The certificate can be used for a period of time after which it must be changed – think of it as a credit card with an expiration date.
  - The central authority does not have to be online all the time – it may not be online at all.

- **The main job of the certification authority is to bind a public key to someone's identity**
  - Note however that the name of the user need not be in the certificate.
  - Instead, the certificate may only state that the user is over 18 years old or has some other kind of rights – this may be useful for anonymity.

- **When Alice has her private key compromised, she is in the same position as losing her credit card:** she must go to the authority and get a new certificate; same after expiration time of her certificate.
Certificate Authority (CA)

\[ C_A = E_{KR_{auth}}[\text{Time}_1, ID_A, KU_a] \]

\[ C_B = E_{KR_{auth}}[\text{Time}_2, ID_B, KU_b] \]

A

B

(1) \( C_A \)

(2) \( C_B \)
A standard for certificates: X.509

- To avoid having different types of certificates for different users, standard X.509 has been issued for the format of certificates – widely used over the Internet.
- At its core, X.509 is a way to describe certificates (see the table on the next slide) – for detailed information see the standard or RFC 2459.
- Example: if Bob works in the loan department of the Money Bank in the US, his X.500 (a series of recommendations) address could be /C=US/O=Money Bank/OU=Loan/CN=Bob, where C is for country, O for organization, OU for organizational unit, CN for common name.
A standard for certificates: X.509

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Which version of X.509</td>
</tr>
<tr>
<td>Serial number</td>
<td>This number plus the CA’s name uniquely identifies the certificate</td>
</tr>
<tr>
<td>Signature algorithm</td>
<td>The algorithm used to sign the certificate</td>
</tr>
<tr>
<td>Issuer</td>
<td>X.500 name of the CA</td>
</tr>
<tr>
<td>Validity period</td>
<td>The starting and ending times of the validity period</td>
</tr>
<tr>
<td>Subject name</td>
<td>The entity whose key is being certified</td>
</tr>
<tr>
<td>Public key</td>
<td>The subject’s public key and the ID of the algorithm using it</td>
</tr>
<tr>
<td>Issuer ID</td>
<td>An optional ID uniquely identifying the certificate’s issuer</td>
</tr>
<tr>
<td>Subject ID</td>
<td>An optional ID uniquely identifying the certificate’s subject</td>
</tr>
<tr>
<td>Extensions</td>
<td>Many extensions have been defined</td>
</tr>
<tr>
<td>Signature</td>
<td>The certificate’s signature (signed by the CA’s private key)</td>
</tr>
</tbody>
</table>
Public key infrastructures (PKI)

- One single CA issuing all the world’s certificates is clearly infeasible
- Several CAs all run by the same organization (which one?) using the same private key to issue certificates introduces the real problem of key leakage – this would ruin the whole world’s e-security
- Proposed solution (still to be standardized): **Public Key Infrastructure (PKI)**
  - Has multiple components, including users, CAs, certificates, directories
  - Give here just a simple form of PKI as a hierarchy of CAs
  - In our example we show only three levels, but in practice there could many more
  - On the top of the hierarchy is the top-level CA (the root)
  - The root certifies second-level CAs that we call RAs (Regional Authorities)
  - RAs certify the real CAs which issue X.509 certificates to individuals and institutions
Public key infrastructures

(a) RA 1
   - CA 1
   - CA 2
   - CA 3

(b) RA 2
   - CA 4
   - CA 5

RA 2 is approved. Its public key is 47383AE349...
Root's signature

RA 2's signature

CA 5 is approved. Its public key is 6384AF863B...

April 11, 2006
Using PKI

- Alice finds Bob’s certificate signed by CA5 – Alice has never heard of CA5
- Alice asks CA5’s certificate: CA5 is certified by RA2
- Alice asks RA2’s certificate: RA2 is certified by the root: everything on the chain is legitimate
- **Question**: How does Alice finds root’s public key?
- **Answer**: it is assumed that everybody knows root’s public key
  - The browser may have it built in
- Better implementation: Bob collects himself the certificates of CA5 and RA2 and makes them available to save Alice the trouble
  - Alice does not need to contact anybody now: all the certificates are signed and she can detect any tampering
- **Problem**: Who is running the root?
- **Solution**: have more than one root, each with its own RAs and CAs
  - Modern browsers come preloaded with the public keys for over 100 roots (**trust anchors**)
- **Problem**: The user should trust the browser vendor to make wise choices and not simply approve all trust anchors willing to pay the inclusion fee
- **Solution**: Most browsers allow users to inspect trust anchors and remove any that may seem shady
- **Problem**: Inconvenient for each user to store his certification path and all certificates
- **Possible solution**: Have the DNS return Bob’s IP address together with all his certificates
Revocation of certificates

- Certificates need to be revoked sometimes: user’s private key has been compromised or maybe the user has abused the certificate somehow or even worse, the CA’s private key has been compromised

- CA periodically issue a Certificate Revocation List (CRL) giving the serial numbers of all certificates it revokes

- **Problem**: Alice should always check the CRL to make sure Bob’s certificate has not been revoked
  - Certificates could also be reinstated
  - This problem eliminates one of the best properties of certificates, that one does not have to constantly contact the CA
  - Advisable to keep a local copy of the CRL