Computer Security (Sheet #4)

Marius Gavrilescu

1. Here we assume that the signature function is essentially unknown to us (because we can't find the key nor make any assumptions about the structure of the function): thus the only attacks we are able to employ are hash collisions, where we have a message-signature pair, and we find a second message that we know to have the same signature.

   Preimage reistance is not important for security in this situation, because we always know the input to the hash. As long as we do not know anything lack of preimage resistance does not enable any attacks.

   Lack of 2nd-preimage resistance means that given a message, we can find a second message with the same hash (and thus same signature). Thus this enables EF under KMA: given a message-signature pair, we can find a second message with the same signature (so we just forged a signature for the second message).

   Under CMA, we have UF: If we want to sign m, find a message m' that has the same hash and ask the oracle to sign m'. The result will be a valid signature for m.

   Lack of collision resistance means we can find a pair of messages that hash to the same thing. This means that if we know the signature for one of them, we also know the signature for the other one. This only enables EF under CMA: to forge a signature for one message ask the oracle to sign the other one.

2. We know for a valid signature s can't be a multiple of $p-1$ and $g^m = g^{xr+sk+z(p-1)} = (g^x)^r(g^k)^s(g^{p-1})^z \equiv (g^x)^r(g^k)^s \equiv y^r r^s (\mathrm{mod}\, p)$.

   So the validation function will return True for a valid signature.

   Take $r = yg^u$ and $s \equiv -yg^u - 1 (\mathrm{mod}\, p)$. Then $g^m = y^{yg^u}(yg^u)^s = (y^{yg^u+s})(g^u)^s \equiv (g^{p-1})^{\cdots} g^{us} \equiv g^{us} (\mathrm{mod}\, p)$, which is a signature for the message us.

   If the s chosen above would be 0 mod p-1, then just pick another u and repeat the process until we get a valid p.

   This is an EF attack on ElGamal.

   It can be prevented by requiring a message to have a particular structure (in which case us might be unlikely to be a valid message) or by signing a hash of the message (thus preventing the attacker from knowing what message they signed, as long as the hash is preimage resistant).

3. Dolev-Yao attacker can stand between Alice and Bob, intercept the second message and substitute their own public key. Then the attacker will be able to find the session key, which means the attacker can read any message, can inject their own messages, can drop any in-flight message, and can replace any in-flight message with one of their own. There is no authentication, no confidentiality, no integrity, and replays are easy.

4. Example attack, with intruder standing between Alice and Bob:

$$A \to I_B : \texttt{Alice} \tag{1}$$
$$I_B \to A : \texttt{Bob} \tag{2}$$
$$A \to I_B : E_{k_{BA}}(nonce) \tag{3}$$
$$I_B \to A : E_{k_{AB}}(nonce) \tag{4}$$
$$A \to I_B : E_{k_{AB}}(nonce + 1) \tag{5}$$
$$I_B \to A : E_{k_{BA}}(nonce + 1) \tag{6}$$

   This works if $k_{AB} = k_{BA}$, and requires Alice to not care about interleaving of messages.

   We therefore should never have the same key for both directions of unidirectional authentication.

5. EKE is successful because the attacker cannot easily know if a guess for the password is right. Assuming the attacker believes the password is p, they compute $pk_E = D_{h(p)}(E_{h(password)}(pk_E))$. The attacker can also compute $h(nonce) = D_{h(p)}(E_{h(password)}(h(nonce)))$. However, there is no way to know that either pk_E or $h(nonce)$ is right, because we cannot decrypt the second message to get the nonce. Thus we can only check if our guess is right by brute-forcing the nonce (given a guess n for the nonce, compute $E_{pk_E}(n)$ and compare against the second message.

   So for each guess for the password we also need to guess the nonce, which greatly increases the key space: we need to brute-force a 2^{84} bit key, which takes 2^{83} work on average.

6. If the ticket does not contain the name of the requester, Mallory can ask the KDC for a shared key between Mallory and Bob, then can give the ticket to Bob while claiming to be Alice. Now Mallory and Bob will be able to communicate securely, but Bob has no way of knowing Mallory's identity.

   If the response from the KDC did not contain the name of the ticket's destination (Bob in our example) then Mallory can intercept Alice's request to the KDC substituting his own name in place of Bob, then the KDC's reply will have a ticket encrypted with Mallory's key, so when Alice sends the ticket to Bob Mallory can intercept it and know the shared key (and thus be able to pretend to be Bob).

7. Alice might have disclosed her password to Mallory, who then authenticated as Alice and committed fraud.

   Alice's computer might be infected with a keylogger that recorded the password as it was typed and relayed it to Mallory.

   Alice might have a short, insecure, easily guessed password.

   The bank might have failed to keep the private key to their certificate secure, thus allowing Mallory to MitM the connection between Alice and the bank.

   The bank might have not rate-limited login attempts and therefore allowed Mallory to brute-force Alice's password using (for example) a dictionary attack.

   The bank might be victim to old-fashioned non-technological fraud; for example a rogue employee might have stolen the funds.

   The CA might have signed Mallory's certificate (accidentally or intentionally), and Alice was (unwittingly) logging into Mallory's fake bank website (thus giving Mallory the password).

   The CA might have failed to publish an updated CRL after the bank informed the CA of a compromised certificate, thus allowing Mallory to use that certificate to impersonate the bank's website.