Massachusetts Institute of Technology
6.857: Network and Computer Security
Professors Srini Devadas and Nickolai Zeldovich

Handout 4
March 29, 2010
Due: April 12, 2010

Problem Set 4

This problem set is due *online,* at `https://sec.csail.mit.edu/` on *Monday, April 12* by 11:59pm. No late submissions will be accepted.

You are to work on this problem set with your assigned group of three or four people. You should have received an email with your group assignment for this problem set. If not, please email `6.857-tas@mit.edu`. Be sure that all group members can explain the solutions. See Handout 1 (*Course Information*) for our policy on collaboration.

Homework must be submitted electronically! Each problem answer must appear on a separate page. Mark the top of each page with your group member names, the course number (6.857), the problem set number and question, and the date. We have provided templates for LaTeX and Microsoft Word on the course website (see the *Resources* page).

Grading: The first problem is worth 20 points. The other two problems are each worth 10 points.

With the authors' permission, we will distribute our favorite solution to each problem as the "official" solution—this is your chance to become famous! If you do not wish for your homework to be used as an official solution, or if you wish that it only be used anonymously, please note this in your profile on the homework submission website.

Problem 3-1. ElGamal Signatures. Recall the ElGamal signature scheme:

- **Key generation:** the signer chooses a random large prime p, random generator $g \in \mathbb{Z}_p^*$, and random $x \in \mathbb{Z}_p^*$, and computes $y = g^x \bmod p$. The public key is (p, g, y). The secret key is x.
- **Signing:** to sign a value $m \in \{0, \ldots, p-2\}$, the signer chooses a random $k \in \mathbb{Z}_{p-1}^*$; that is, a $k \in \{1, \ldots, p-2\}$ such that $\gcd(k, p-1) = 1$. The signer computes $r = g^k \bmod p$, and

  $$s = (m - xr)k^{-1} \bmod (p-1).$$

  The signer repeats these steps until $s \neq 0$. The signature for m is the pair (r, s).
- **Verification:** For value m with signature (r, s), the verifier checks that $0 < r < p$ and $0 < s < p-1$, and also checks that $g^m = y^r r^s \bmod p$. If all conditions are true, the verifier accepts, otherwise he rejects.

This scheme is believed to be secure, but there are many precautions that must be taken in its implementation.

(a) Show that the signer *must* keep k secret, in the sense that if k is known, an attacker can learn the signer's secret key.

(b) Show that if the same value of k is used to sign two different messages, the secret key can also be found.

(c) In contexts where it is difficult or impossible to access true random numbers, people sometime rely on so-called *pseudo-random number generators*. These are *deterministic* algorithms that start with a *random seed*. Their output is a sequence of numbers computed deterministically from the seed.

One simple and popular generator is called the *linear congruential generator*. Used in this context, there are two public values $a, b \in \mathbb{Z}_p^*$. The initial seed is a random, secret $k_0 \bmod p$. When an ith value is to be signed, the value $k_i = a \cdot k_{i-1} + b \bmod p - 1$ is computed and used for k.

Is it safe for the signer to use such a generator for ElGamal signatures?

(d) Show that the verifier must check that $r < p$, otherwise it is possible to forge signatures many values m' after seeing a single signature for some arbitrary value m. *Hint:* given the valid signature (r, s), the forged signature (r', s') should be such that $r' = r \bmod p$.

Problem 3-2. Hardware Attacks.

Consider a smart card whose public key is $(n, e, X^e \bmod n)$ (where (n, e) is an RSA public key and X is a random element of $\mathbb{Z}_n^*$). The private key of the smart card is X.

Whenever the smart card needs random numbers, it generates them using a pseudo random generator which starts with a random seed S stored on the card. The card also has its own battery. When the battery is disconnected, the current state of the smart card is lost, and it restarts from scratch. The values (n, e, X, S) are stored in non-volatile memory, which does not require battery power to be maintained.

(a) Consider the following authentication protocol run on the card.

1. The smart card selects an element R in $\mathbb{Z}_n^*$ at random, and sends $R^e \bmod n$ to the server.
2. The server picks a random bit b and sends it to the smart card.
3. The smart card computes $Y = RX^b \bmod n$ and sends it to the server.
4. The server accepts if $Y^e = R^e(X^e)^b \bmod n$.

This protocol is repeated 50 times to provide the server with an acceptably low error probability. Explain how an attacker with temporary access to the smart card could recover the secret X.

Ben Bitdiddle has been placed in charge of designing an authentication system for a small, low-powered handheld PDA device, so that only the legitimate owner of the device can use it. The device receives user input solely through a touch-sensitive screen, which the user draws on with a special stylus.

Ben's idea works as follows: the user is assigned a random sequence of m symbols from a fixed alphabet Σ. In order to authenticate himself to the device, the user draws the sequence of symbols on the screen. For each symbol, the device employs a sophisticated, expensive `match` algorithm to determine whether the drawn symbol matches the correct one. Pseudocode for the authentication algorithm is as follows:

```
for i = 1 to m:
   if(!match(drawn_symbol[i], secret_symbol[i]))
      return false   // authentication failed
return true          // authentication succeeded
```

(b) Explain how Ben's method is subject to a side-channel attack. Estimate how much effort it would take an attacker to successfully authenticate, versus how much effort it would take to mount a brute-force attack.

(c) Suggest a fix to Ben's authentication scheme, without changing the basic setup.

Problem 3-3. Public Key Infrastructure

(Compare and constrast X.509 certificates and SPKI/SDSI certificates.)

An X.509 certificate roughly says: "I am the principal with distinguished name /A/B/C and public key K1, and I hereby certify that the principal with name /A/B/C/D has public key K2. (Certificate issued mm/dd/yyyy; expires mm/dd/yyyy.)(signed by K1)"

A (simple) SPKI/SDSI certificate roughly says: "I am the principal with public key K1, and I hereby assert that if you know me by some name X, then you should know the principal with public key K2 by name X/D. (Certificate issued mm/dd/yyyy; expires mm/dd/yyyy.) (signed by K1)"

To get going, SPKI/SDSI automatically allows hash(K) as a name for the principal with public key K; so names in general take the form hash(K)/A/B/C, etc. There is no single "root" identified. Names in general are relative to the starting point K.

Compare and contrast these two forms of certificates as primitive building blocks for a public-key infrastructure (PKI). Discuss as many relevant aspects as you can, include signing documents, certificate chains, flexibility when organizations change or merge, keys are compromised, etc.