CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 15
October 21, 2013

Hash from Cryptosystem

Authentication Using Passwords

- Authentication problem
- Passwords authentication schemes
- Secure password storage
- Dictionary attacks

Chinese Remainder Theorem

Quadratic Residues, Squares, and Square Roots

- Square roots modulo an odd prime p
- Square roots modulo the product of two odd primes

Hash from Cryptosystem

Building hash functions from cryptosystems

We've already seen several cryptographic hash functions as well as methods for making new hash functions from old.

We describe a way to make a hash function from a symmetric cryptosystem with encryption function $E_k(b)$.

Assume the key and block lengths are the same. (This rules out DES but not AES with 128-bit keys.)

The construction

Let m be a message of arbitrary length. Here's how to compute $H(m)$.

- Pad m appropriately and divide it into block lengths appropriate for the cryptosystem.
- Compute the following state sequence:

$$\begin{aligned} s_0 &= \mathrm{I}V \\ s_1 &= f(s_0, m_1) \\ &\vdots \\ s_t &= f(s_{t-1}, m_t). \end{aligned}$$

- Define $H(m) = s_t$.

IV is an initial vector and f is a function built from E.

Possible state transition functions $f(s, m)$

Some possibilities for f are

$$f_1(s, m) = E_s(m) \oplus m$$
$$f_2(s, m) = E_s(m) \oplus m \oplus s$$
$$f_3(s, m) = E_s(m \oplus s) \oplus m$$
$$f_4(s, m) = E_s(m \oplus s) \oplus m \oplus s$$

You should think about why these particular functions do or do not lead to a strong collision-free hash function.

A bad state transition function

For example, if $t = 1$ and $f = f_1$, then

$$H(m) = f_1(\mathrm{I}V, m) = E_{\mathrm{I}V}(m) \oplus m.$$

$E_{\mathrm{I}V}$ itself is one-to-one (since it's an encryption function), but what can we say about $H_1(m)$?

Indeed, if bad luck would have it that E_{IV} is the identity function, then $H(m) = 0$ for all m, and all pairs of message blocks collide!

Authentication Using Passwords

The authentication problem

The *authentication problem* is to identify whom one is communicating with.

For example, if Alice and Bob are communicating over a network, then Bob would like to know that he is talking to Alice and not to someone else on the network.

Knowing the IP address or URL is not adequate since Mallory might be in control of intermediate routers and name servers.

As with signature schemes, we need some way to differentiate the real Alice from other users of the network.

Possible authentication factors

Alice can be authenticated in one of three ways:

1. By something she knows;
2. By something she possesses;
3. By something she is.

Examples:

1. A secret password;
2. A smart card;
3. Biometric data such as a fingerprint.

Passwords

Assume that Alice possess some secret that is not known to anyone else. She authenticates herself by proving that she knows the secret.

Password mechanisms are widely used for authentication.

In the usual form, Alice authenticates herself by sending her password to Bob.

Bob checks that it matches Alice's password and grants access.

This is the scheme that is used for local logins to a computer and is also used for remote authentication on many web sites.

Weaknesses of password schemes

Password schemes have two major security weaknesses.

1. Passwords may be exposed to Eve when being used.
2. After Alice authenticates herself to Bob, Bob can use Alice's password to impersonate Alice.

Password exposure

Passwords sent over the network in the clear are exposed to various kinds of eavesdropping, ranging from ethernet packet sniffers on the LAN to corrupt ISP's and routers along the way.

The threat of password capture in this way is so great that one should *never* send a password over the internet in the clear.

Some precautions

Users of the old insecure Unix tools should switch to secure replacements such as ssh, slogin, and scp, or kerberized versions of telnet and ftp.

Web sites requiring user logins generally use the TSL/SSL (Transport Layer Security/Secure Socket Layer) protocol to encrypt the connection, making it safe to transmit passwords to the site, but some do not.

Depending on how your browser is configured, it will warn you whenever you attempt to send unencrypted data back to the server.

Password propagation

After Alice's password reaches the server, it is no longer the case that only she knows her password.

Now the server knows it, too!

This is no problem if Alice only uses her password to log into that *that particular* server.

However, if she uses the same password for other web sites, the first server can impersonate Alice to any other web site where Alice uses the same password.

Multiple web sites

Users these days typically have accounts with dozens or hundreds of different web sites.

The temptation is strong to use the same username-password pairs on all sites so that they can be remembered.

But that means that anyone with access to the password database on one site can log into Alice's account on any of the other sites.

Typically different sites have very differing sensitivity of the data they protect.

An on-line shopping site may only be protecting a customer's shopping cart, whereas a banking site allows access to a customer's bank account.

Password policy advice

My advice is to use a different password for each account.

Of course, nobody can keep dozens of different passwords straight, so the downside of my suggestion is that the passwords must be written down and kept safe, or stored in a properly-protected password vault.

If the primary copy gets lost or compromised, then one should have a backup copy so that one can go to all of the sites ASAP and change the passwords (and learn if the site has been compromised).

The real problem with simple password schemes is that Alice is required to send her secrets to other parties in order to use them. We will later explore authentication schemes that do not require this.

Secure password storage

Another issue with traditional password authentication schemes is the need to store the passwords on the server for later verification.

- The file in which passwords are store is highly sensitive.
- Operating system protections can (and should) be used to protect it, but they are not really sufficient.
- Legitimate sysadmins might use passwords found there to log into users' accounts at other sites.
- Hackers who manage to break into the computer and obtain root privileges can do the same thing.
- Finally, backup copies may not be subject to the same system protections, so someone with access to a backup device could read everybody's password from it.

Storing encrypted passwords

Rather than store passwords in the clear, it is usual to store "encrypted" passwords.

That is, the hash value of the password under some cryptographic hash function is stored instead of the password itself.

Using encrypted passwords

The authentication function

- takes the cleartext password from the user,
- computes its hash value,
- and checks that the computed and stored hashed values match.

Since the password does not contain the actual password, and it is computationally difficult to invert a cryptographic hash function, knowledge of the hash value does not allow an attacker to easily find the password.

Dictionary attacks on encrypted passwords

Access to an encrypted password file opens up the possibility of a *dictionary attack*.

Many users choose weak passwords—words that appear in an English dictionary or in other available sources of text.

If one has access to the password hashes of legitimate users on the computer (such as is contained in `/etc/passwd` on Unix), an attacker can hash every word in the dictionary and then look for matches with the password file entries.

Harm from dictionary attacks

A dictionary attack is quite likely to succeed in compromising at least a few accounts on a typical system.

Even one compromised account is enough to allow the hacker to log into the system as a legitimate user, from which other kinds of attacks are possible that cannot be carried out from the outside.

Salt

Adding salt is a way to make dictionary attacks more expensive.

- *Salt* is a random number that is stored along with the hashed password in the password file.
- The hash function takes two arguments, the password and salt, and produces a hash value.
- Because the salt is stored (in the clear) in the password file, the user's password can be easily verified.
- The same password hashes differently depending on the salt.
- A successful dictionary attack now has to encrypt the entire dictionary with every possible salt value (or at least with every salt value that appears in the password file being attacked).
- This increases the cost of the attack by orders of magnitude.

Chinese Remainder Theorem

Systems of congruence equations

Theorem (Chinese remainder theorem)

Let $n_1, n_2, \ldots, n_k$ be positive pairwise relatively-prime integers[1]*, let $n = \prod_{i=1}^{k} n_i$, and let $a_i \in \mathbf{Z}_{n_i}$ for $i = 1, \ldots, k$. Consider the system of congruence equations with unknown x:*

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned} \tag{1}$$

(1) has a unique solution $x \in \mathbf{Z}_n$.

[1] This means that $\gcd(n_i, n_j) = 1$ for all $1 \leq i < j \leq k$.

How to solve congruence equations

To solve for x, let

$$N_i = n/n_i = \underbrace{n_1 n_2 \ldots n_{i-1}} \cdot \underbrace{n_{i+1} \ldots n_k},$$

and compute $M_i = N_i^{-1} \bmod n_i$, for $1 \leq i \leq k$.

$N_i^{-1} \pmod{n_i}$ exists since $\gcd(N_i, n_i) = 1$. (Why?)
We can compute N_i^{-1} by solving the associated Diophantine equation as described in Lecture 10.

The solution to (1) is

$$x = (\sum_{i=1}^{k} a_i M_i N_i) \bmod n \qquad (2)$$

Correctness

Lemma

$$M_j N_j \equiv \begin{cases} 1 \pmod{n_i} & \text{if } j = i; \\ 0 \pmod{n_i} & \text{if } j \neq i. \end{cases}$$

Proof.

$M_i N_i \equiv 1 \pmod{n_i}$ since $M_i = N_i^{-1} \bmod n_i$.
If $j \neq i$, then $M_j N_j \equiv 0 \pmod{n_i}$ since $n_i | N_j$. □

It follows from the lemma and the fact that $n_i \mid n$ that

$$x \equiv \sum_{i=1}^{k} a_i M_i N_i \equiv a_i \pmod{n_i} \tag{3}$$

for all $1 \leq i \leq k$, establishing that (2) is a solution of (1).

Uniqueness

To see that the solution is unique in $\mathbf{Z}_n$, let $\chi : \mathbf{Z}_n \to \mathbf{Z}_{n_1} \times \ldots \times \mathbf{Z}_{n_k}$ be the mapping

$$x \mapsto (x \bmod n_1, \ldots, x \bmod n_k).$$

χ is a surjection[2] since $\chi(x) = (a_1, \ldots, a_k)$ iff x satisfies (1).

Since also $|\mathbf{Z}_n| = |\mathbf{Z}_{n_1} \times \ldots \times \mathbf{Z}_{n_k}|$, χ is a bijection, and there is only one solution to (1) in $\mathbf{Z}_n$.

[2] A *surjection* is an onto function.

An alternative proof of uniqueness

A less slick but more direct way of seeing uniqueness is to suppose that $x = u$ and $x = v$ are both solutions to (1).

Then $u \equiv v \pmod{n_i}$, so $n_i | (u - v)$ for all i.

By the pairwise relatively prime condition on the n_i, it follows that $n | (u - v)$, so $u \equiv v \pmod{n}$. Hence, the solution is unique in $\mathbf{Z}_n$.

Quadratic Residues, Squares, and Square Roots

Square roots in $\mathbf{Z}_n^*$

Recall from lecture 13 that to find points on an elliptic curve requires solving the equation

$$y^2 = x^3 + ax + b$$

for $y \pmod{p}$, and that requires computing square roots in $\mathbf{Z}_p^*$.

Squares and square roots have several other cryptographic applications as well.

Today, we take a brief tour of the theory of *quadratic resides*.

Quadratic residues modulo n

An integer b is a *square root* of a modulo n if

$$b^2 \equiv a \pmod{n}.$$

An integer a is a *quadratic residue (or perfect square)* modulo n if it has a square root modulo n.

Quadratic residues in $\mathbf{Z}_n^*$

If $a, b \in \mathbf{Z}_n$ and $b^2 \equiv a \pmod{n}$, then

$$b \in \mathbf{Z}_n^* \text{ iff } a \in \mathbf{Z}_n^*.$$

Why? Because

$$\gcd(b, n) = 1 \text{ iff } \gcd(a, n) = 1$$

This follows from the fact that $b^2 = a + un$ for some u, so if p is a prime divisor of n, then

$$p \mid b \text{ iff } p \mid a.$$

Assume that all quadratic residues and square roots are in $\mathbf{Z}_n^*$ unless stated otherwise.

QR_n and QNR_n

We partition $\mathbf{Z}_n^*$ into two parts.

$$\mathrm{QR}_n = \{a \in \mathbf{Z}_n^* \mid a \text{ is a quadratic residue modulo } n\}.$$
$$\mathrm{QNR}_n = \mathbf{Z}_n^* - \mathrm{QR}_n.$$

QR_n is the *set of quadratic residues* modulo n.

QNR_n is the *set of quadratic non-residues* modulo n.

For $a \in \mathrm{QR}_n$, we sometimes write

$$\sqrt{a} = \{b \in \mathbf{Z}_n^* \mid b^2 \equiv a \pmod{n}\},$$

the *set of square roots* of a modulo n.

Quadratic residues in $\mathbf{Z}_{15}^*$

The following table shows all elements of $\mathbf{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$ and their squares.

b	$b^2 \bmod 15$
1	1
2	4
4	1
7	4
$8 = -7$	4
$11 = -4$	1
$13 = -2$	4
$14 = -1$	1

Thus, $\mathrm{QR}_{15} = \{1, 4\}$ and $\mathrm{QNR}_{15} = \{2, 7, 8, 11, 13, 14\}$.

Quadratic residues modulo an odd prime p

Fact

For an odd prime p,

- *Every $a \in QR_p$ has exactly two square roots in $\mathbf{Z}_p^*$;*
- *Exactly 1/2 of the elements of $\mathbf{Z}_p^*$ are quadratic residues.*

In other words, if $a \in \mathrm{QR}_p$,

$$|\sqrt{a}| = 2.$$

$$|\mathrm{QR}_n| = |\mathbf{Z}_p^*|/2 = \frac{p-1}{2}.$$

Quadratic residues in $\mathbf{Z}_{11}^*$

The following table shows all elements $b \in \mathbf{Z}_{11}^*$ and their squares.

b	$b^2 \bmod 11$
1	1
2	4
3	9
4	5
5	3

b	$-b$	$b^2 \bmod 11$
6	-5	3
7	-4	5
8	-3	9
9	-2	4
10	-1	1

Thus, $\mathrm{QR}_{11} = \{1, 3, 4, 5, 9\}$ and $\mathrm{QNR}_{11} = \{2, 6, 7, 8, 10\}$.

Proof that $|\sqrt{a}| = 2$ modulo an odd prime p

Let $a \in \mathrm{QR}_p$.

- It must have a square root $b \in \mathbf{Z}_p^*$.
- $(-b)^2 \equiv b^2 \equiv a \pmod{p}$, so $-b \in \sqrt{a}$.
- Moreover, $b \not\equiv -b \pmod{p}$ since $p \nmid 2b$, so $|\sqrt{a}| \geq 2$.
- Now suppose $c \in \sqrt{a}$. Then $c^2 \equiv a \equiv b^2 \pmod{p}$.
- Hence, $p \,|\, c^2 - b^2 = (c-b)(c+b)$.
- Since p is prime, then either $p \,|\, (c-b)$ or $p \,|\, (c+b)$ (or both).
- If $p \,|\, (c-b)$, then $c \equiv b \pmod{p}$.
- If $p \,|\, (c+b)$, then $c \equiv -b \pmod{p}$.
- Hence, $c = \pm b$, so $\sqrt{a} = \{b, -b\}$, and $|\sqrt{a}| = 2$.

Proof that half the elements of $\mathbf{Z}_p^*$ are in QR_p

- Each $b \in \mathbf{Z}_p^*$ is the square root of exactly one element of QR_p.
- The mapping $b \mapsto b^2 \bmod p$ is a 2-to-1 mapping from $\mathbf{Z}_p^*$ to QR_p.
- Therefore, $|\mathrm{QR}_p| = \frac{1}{2}|\mathbf{Z}_p^*|$ as desired.

Quadratic residues modulo *pq*

We now turn to the case where $n = pq$ is the product of two distinct odd primes.

Fact

Let $n = pq$ for p, q distinct odd primes.

- *Every $a \in QR_n$ has exactly four square roots in $\mathbf{Z}_n^*$;*
- *Exactly 1/4 of the elements of $\mathbf{Z}_n^*$ are quadratic residues.*

In other words, if $a \in \mathrm{QR}_n$,

$$|\sqrt{a}| = 4.$$

$$|\mathrm{QR}_n| = |\mathbf{Z}_n^*|/4 = \frac{(p-1)(q-1)}{4}.$$

Proof sketch

- Let $a \in \mathrm{QR}_n$. Then $a \in \mathrm{QR}_p$ and $a \in \mathrm{QR}_q$.
- There are numbers $b_p \in \mathrm{QR}_p$ and $b_q \in \mathrm{QR}_q$ such that
  - $\sqrt{a} \pmod{p} = \{\pm b_p\}$, and
  - $\sqrt{a} \pmod{q} = \{\pm b_q\}$.
- Each pair (x, y) with $x \in \{\pm b_p\}$ and $y \in \{\pm b_q\}$ can be combined to yield a distinct element $b_{x,y}$ in $\sqrt{a} \pmod{n}$.[3]
- Hence, $|\sqrt{a} \pmod{n}| = 4$, and $|\mathrm{QR}_n| = |\mathbf{Z}_n^*|/4$.

[3] To find $b_{x,y}$ from x and y requires use of the Chinese Remainder theorem.