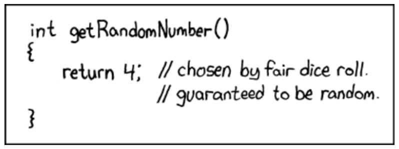
CPSC 418/MATH 318 Introduction to Cryptography

More on Digital Signatures, El Gamal Signature Scheme, Random Number Generation, Authentication

Renate Scheidler

Department of Mathematics & Statistics
Department of Computer Science
University of Calgary

Week 11



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

Outline

1. Security of Signatures
2. ElGamal Signature Scheme
3. Cryptography in Real Life
   - Random Number Generation
4. Where are we at?
5. Authentication

Digital Signatures: Recap

Definition 1 (Digital signature)

A means for data origin authentication and non-repudiation that should have two properties:

1. Only the sender can produce their signature.
2. *Anyone* should be easily able to verify the validity of the signature.

Property 1 provides authentication of the entity where the data originated from.

Properties 1 & 2 provide non-repudiation (resolution of disputes).

Provable Security of Signatures

In practice, signature schemes must be resistant to active attacks. We need the equivalent of IND-CCA2 for signatures.

Definition 2 (GMR-security)

A signature scheme is said to be *GMR-secure* if it is existentially unforgeable by a computationally bounded adversary who can mount an adaptive chosen-message attack.

In other words, an adversary who can obtain signatures of any messages of her own choosing from the legitimate signer is unable to produce a valid signature of any new message (for which she has not already requested and obtained a signature) in polynomial time.

GMR stands for *Goldwasser-Micali-Rivest*.

GMR-Secure Versions of RSA

Example 3

RSA-PSS (Probabilistic Signature Scheme), a digital signature analogue of OAEP, is GMR-secure in the random oracle model (ROM) assuming that the RSA problem (computing e-th roots modulo n) is hard.

Example 4

RSA with *full-domain hash* — use RSA signatures as usual, signing $H(M)$, but select the hash function H such that $0 \le H(M) < n$ (n is the RSA modulus) for all messages M (see Section 8.5 of Stinson-Paterson).

- Called full-domain because the messages signed are taken from the entire range of possible RSA blocks as opposed to a smaller subrange.
- Also GMR-secure under same assumption as above.

The El Gamal Signature Scheme

The El Gamal signature scheme is a variation of the El Gamal PKC (same 1985 paper). Security considerations are the same.

Alice produces her public and private keys as follows:

1. Selects a large prime p and a primitive root g of p.
2. Randomly selects x such that $0 < x < p-1$ and computes $y \equiv g^x \pmod{p}$.

Public key: (p, g, y)
Private key: $\{x\}$

Alice also fixes a public cryptographic hash function $H : \{0,1\}^* \mapsto \mathbb{Z}_{p-1}$.

Signing and Verifying

Alice signs a message $M \in \{0,1\}^*$ as follows:

1. Selects a random integer $k \in \mathbb{Z}_{p-1}^*$.
2. Computes $r \equiv g^k \pmod{p}$, $0 \le r < p$.
3. Solves $ks \equiv H(M\|r) - xr \pmod{p-1}$ for $s \in \mathbb{Z}_{p-1}^*$
4. Alice's signature is the pair (r, s).

Bob verifies A's signature (r, s) as follows:

1. Obtains Alice's authentic public key (p, g, y).
2. Computes $v_1 \equiv y^r r^s \pmod{p}$ and $v_2 \equiv g^{H(M\|r)} \pmod{p}$.
3. Accepts the signature if and only if $r < p$ and $v_1 = v_2$.

Proof of Correctness for Verification

Proof of correctness.

Note that $ks + rx = H(M\|r) + m(p-1)$ for some integer m. If the signature (r, s) to message M is valid, then

$$\begin{aligned} v_1 &\equiv y^r r^s \\ &\equiv (g^x)^r (g^k)^s \\ &\equiv g^{xr+ks} \\ &\equiv g^{H(M\|r)+m(p-1)} \\ &\equiv g^{H(M\|r)} (g^m)^{p-1} \\ &\equiv v_2 \pmod{p}, \end{aligned}$$

where we use that $(g^m)^{p-1} \equiv 1 \pmod{p}$ by Fermat's Little Theorem. □

Solving General Linear Congruences

For signature generation step 3, we need to solve the congruence

$$ks \equiv H(M\|r) - xr \pmod{p-1} \quad \text{for} \quad s \in \mathbb{Z}_{p-1}^* \ .$$

More generally, we want to solve a linear congruence of the form

$$aX \equiv b \pmod{m}$$

for $X \in \mathbb{Z}_m^*$, with $m \in \mathbb{N}$, $a \in \mathbb{Z}_m^*$ and $b \in \mathbb{Z}_m$.

We already saw how to do this for $b = 1$; that's just finding modular inverses.

To solve $aX \equiv b \pmod{m}$ for X, first solve $aZ \equiv 1 \pmod{m}$ for Z using the Extended Euclidean Algorithm. Then $X \equiv bZ \pmod{m}$ as

$$aX \equiv a(bZ) \equiv (aZ)b \equiv 1 \cdot b \equiv b \pmod{m} \ .$$

ElGamal Example: Set-Up

Let $p = 467$, and set $g = 2$ which is a primitive root modulo 467.

- Choose the secret key $x = 127$
- Using binary exponentiation, one obtains $y \equiv 2^{127} \equiv 132 \pmod{467}$

So consider an ElGamal user Alice with

- public key $(467, 2, 132)$
- private key $\{127\}$

ElGamal Example: Signature Generation

Suppose Alice wishes to sign the message $M =$ "Hi there".

- She selects $k = 213$; note that $\gcd(213, 466) = 1$.
- Binary exponentiation yields $r \equiv 2^{213} \equiv 29 \pmod{467}$.

Suppose our hash function yields $H(\text{"Hi there"} \| 29) = 100$.

- Alice needs to solve

$$213s \equiv 100 - 127 \cdot 29 \equiv 145 \pmod{466} \ .$$

- First solve $213z \equiv 1 \pmod{466}$ for z using the Extended Euclidean Algorithm, obtaining $z \equiv 431 \pmod{466}$.
- Then $s \equiv 145 \cdot 431 \equiv 51 \pmod{466}$.
- The signature to "Hi there" is $(r, s) = (29, 51)$.

ElGamal Example: Signature Verification

To verify this signature, first note that $r = 29 < 467$. Then compute

$$v_1 \equiv 132^{29} \cdot 29^{51} \equiv 189 \pmod{467}$$

and $v_2 \equiv 2^{100} \equiv 189 \pmod{467}$. So $v_1 = v_2 = 189$.

Existential Forgery Against El Gamal Without Hashing M

Suppose the hash function in El Gamal is omitted:

- Solve $ks \equiv M - rx \pmod{p-1}$ in step 3 of signature generation.
- Verify that $y^r r^s \equiv g^M \pmod{p}$.

Then the following existential forgery attack is possible: put

$$r \equiv gy \pmod{p} \ , \quad s = M = p - 1 - r \ .$$

Then

$$y^r r^s \equiv y^r (gy)^{p-1-r} \equiv y^{r+p-1-r} g^{p-1-r} \equiv y^{p-1} g^M \equiv g^M \pmod{p} \ .$$

This attack extends to many other values of r, s, M (see Stinson-Paterson pp. 317f.)

Hashing M with a pre-image resistant hash function H foils this attack, as finding a "message" M such that $H(M) = p - 1 - r$ requires finding a pre-image of $p - 1 - r$ under H.

Universal Forgery Against El Gamal Without Hashing r

If $H(M\|r)$ is replaced by $H(M)$ in step 3 of the signature generation, then a universal forgery attack is possible.

- More exactly, if an attacker intercepts a signature (r, s) to a message m, he can forge a signature (R, S) to an *arbitrary* message M.
- The resulting R satisfies $0 \leq R \leq p(p-1)$.
- This attack can be foiled by verifying that $r < p$ and rejecting signatures where r exceeds p. Better to include r in the hash though.

See Problem 7 of Assignment 3.

Re-Using Random Numbers in Signature Generation

Suppose the same value of k is used to sign two messages M_1, M_2. Then k and the private key x can be recovered with high probability:

$$\begin{aligned} ks_1 &\equiv H(M_1\|r) - xr \pmod{p-1} \ , \\ ks_2 &\equiv H(M_2\|r) - xr \pmod{p-1} \ , \end{aligned}$$

where $r \equiv g^k \pmod{p}$. Subtracting these two congruences yields

$$k(s_1 - s_2) \equiv H(M_1\|r) - H(M_2\|r) \pmod{p-1} \ .$$

- Can solve for k if $\gcd(s_1 - s_2, p - 1) = 1$.
- Can then solve $xr \equiv H(M_1\|r) - ks_1 \pmod{p-1}$ for x if $\gcd(r, p-1) = 1$.

Security of ElGamal Signatures

El Gamal as presented (i.e. hashing both M and r) is GMR-secure in the ROM assuming that H takes on random values and computing discrete logarithms modulo p is hard.

- Formally, one shows that the DLP reduces to existential forgery, *i.e.* that an algorithm for producing existential forgeries can be used to solve the DLP.

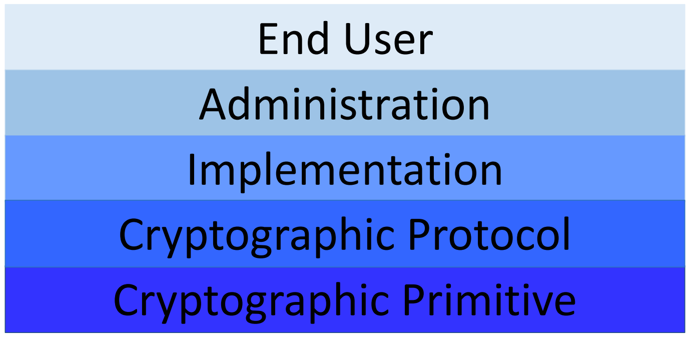
The public parameter g must be chosen verifiably at random (eg. publish PRNG, seed, and algorithm used) in order to ensure that g is a primitive root of p (applies to Diffie-Hellman and El Gamal cryptosystem as well; analogous for RSA).

Other DLP Based Signature Schemes

- Digital Signature Algorithm (DSA) — variation of ElGamal with short signatures, standardized by NIST in 1994 under the name *Digital Signature Standard* (FIPS 186)
- Feige-Fiat-Shamir — security based on computing square roots modulo pq
- Guillou-Quisquater — security based on the RSA problem (computing e-th roots modulo pq)

A description of DSA can be found on the “handouts” page and in Section 8.4.2 of Stinson-Paterson.

Crypto — From Drawing Board to Real Life



Most real-life problems happen at the top three levels.

We need to start thinking about *practical cryptography*!

Random Numbers in Cryptography

There are many uses of **random numbers** in cryptography:

- keys for conventional cryptosystems
- randomized schemes
- public key generation
- key stream for a stream cipher
- One-time values (*nonces*) in authentication protocols to prevent replay

It is critical that these values be

- **statistically random** — independent, uniform distribution
- **unpredictable** — cannot infer future sequence on previous values)

How to Obtain Randomness?

The only source of true randomness is the real world.

- Find a regular but random event and monitor. Examples:
    - radioactive radiation
    - radio noise (white noise)
    - thermal noise in diodes
    - leaky capacitors
    - mercury discharge tubes, etc.
- Need special hardware in general (e.g. radiation counters)
- Can be slow and cumbersome
- Problems of bias or uneven distribution — have to compensate or use noisiest bits from each sample). One possibility: pass data through a cryptographically secure hash function.

Pseudo-Randomness

Published collections of random numbers also exist, but they are too limited and well-known for most uses.

In practice, one uses **pseudo-randomness**.

Definition 1 (Pseudorandom Number/Bit Generator (PRNG, PRBG))

An algorithmic technique to create sequences of statistically random numbers/bits, initialized with a random **seed**.

Example

Linear congruential generator:

$$X_{i+1} = aX_i + c \pmod{m} ,$$

where m is a positive integer (e.g. $m = 2^{31}$), $a \in \mathbb{Z}_m^*$ and $c \in \mathbb{Z}_m$.

- Advantage: outputs long statistically random sequences
- Disadvantage: fails unpredictability — it is too easy to reconstruct entire sequence given only a few values

Cryptographically Secure PRBGs

Definition 2 (Cryptographically secure PRBG (CSPRBG))

Must pass the **next-bit test**: there is no polynomial time algorithm that, on input of the first k bits of an output sequence, can predict the $(k+1)$-st bit with probability significantly greater than $1/2$.

For all practical purposes, a CSPRBG is unpredictable.

Remark: A PRBG is cryptographically secure if and only if it passes the **previous bit test**: there is no polynomial time algorithm that, on input of k bits of an output sequence can predict the preceding bit with probability significantly greater than $1/2$.

More Examples of PRBGs

Simple Examples (see NIST SP-800-90)

Idea: output of a strong hash function or block cipher is statistically random

- $X_i = H(X_{i-1})$ where X_0 is a random seed. Predictable, but good for distilling random bits from another source (whitening).
- $X_i = E_{K_m}(C+1)$ where K_m is a protected master key and C is a counter of some long period. Seems to be computationally infeasible to predict next X_i if K_m is secret.

More Complicated Example: Blum-Blum-Schub PRBG

- Bit parity of $\{X_i\}_{i \geq 0}$ where

$$X_{i+1} \equiv X_i^2 \pmod{n} , \quad X_0 \in \mathbb{Z}_n^* , \quad n = pq .$$

Satisfies the next-bit test under the assumption that the QRP is intractable.

Common Mistakes with PRNGs

The security of a PRNG is determined by the entropy of its seed (which is the bit length if the seed is random). So the seed must have sufficient **entropy** to make the PRNG unpredictable.

The following are all real life (bad) examples!

1. Generating a random 512-bit prime using a 32-bit seed for the random number generator. The entropy of the resulting prime is only 32 bits — easy to exhaustively try all possible seeds.
2. Generating a random 512-bit prime by calling a system PRNG that produces 32-bit random numbers, padding with 0s to 512 bits, and looking for the smallest prime greater than the number. This approach also has only 32 bits of entropy.
3. Instead of padding with zeros, call the system PRNG once and concatenate the resulting 32-bit random number 16 times to obtain a 512-bit number. This still has only 32 bits of entropy.

Another Bad Example — Kerberos 4

Kerberos 4 generates DES session keys by using a PRNG, seeded with a 32-bit value, to produce two 32-bit random numbers.

Problem: only 32 bits of entropy (should be 56)

Bigger problem: seed is the XOR of 5 random 32-bit numbers:

- time of day in seconds since Jan. 1, 1970
- fractional part of the current time in microseconds
- process ID of Kerberos server process
- cumulative count of session keys produced so far
- host id of the machine on which Kerberos is running

Entropy of each of these quantities: between 1 to 20 bits

Thus, Kerberos 4 seed has only 20 **bits of entropy** — it is easy to test all 2^{20} possible values in seconds! (Better in Kerberos 5.)

Possible Fix to Kerberos 4

Compute a hash on the concatenation of the 5 values.

- Every bit of randomness contributes to every bit of the session key
- Successive applications of the hash function will produce further pseudorandom bits (but with no more total entropy than the seed)

See the Internet Engineering Task Force's (IETF) Request For Comments RFC 1750 "Randomness Recommendations for Security" for more information about guidelines for deploying random number generators. Section 6 covers recommendations for software-based strategies for example.

A Final Bad Example — Factoring RSA Moduli

1. Scrounge the internet for lots of RSA public keys with moduli $n_1, n_2, \ldots$
2. Compute $\gcd(n_i, n_j)$ for lots of $i \neq j$

You'd be surprised how many of the moduli you can factor!

Problem: too many people use the same primes, obtained via bad PRNGs.

So be sure to mind your p's and q's !

NIST Recommendations

Moral: The number of bits of entropy must correspond to the overall bit security of the system.

Example: 3072-bit RSA provides 128 bits of security, so the seed material for the PRNG must have at least 128 bits of entropy.

NIST's Recommendations for Security levels (SP 800-57 part 1):

Security level (in bits)	80	112	128	192	256
Hash function size (in bits)	160	224	256	384	512
RSA modulus (in bits)	1024	2048	3072	8192	15360

Security level: key length for block cipher providing equivalent level of difficulty to break

The first two security levels (80 and 112) are now considered insufficient. Levels 3, 4 and 5 (128, 192, 256 bits, respectively) are considered secure.

Were are we at?

Recall cryptographic services:

- Data confidentiality: discussed
- Data integrity: discussed
- Authentication: discussed, more next
- Non-repudiation: discussed
- Access Control: discussed a bit

Recall cryptographic mechanisms:

- Encryption — for confidentiality and limited data integrity: discussed
- Hash functions, Message Authentication Codes (MACs) — for data integrity : discussed
- Digital signatures — for data origin authentication and non-repudiation: discussed
- Authentication protocol — for entity authentication: next

Authentication

Next topic: authentication in practice.

Today, *authentication* is arguably the most important application of cryptography. Three main classifications:

- Message authentication (MACs) — covered
- Data-origin authentication (digital signatures) — covered previously
- Authenticated key establishment — covered next
- Entity authentication (client-server, user-host, process-host) — covered after that

In practice, these are often combined into one protocol (e.g. SSL/TLS).

Authenticity of Keys

Secure communication requires proper mechanisms for managing keys and ensuring their **authenticity**.

Mechanisms for ensuring authenticity of keys:

- A trusted third party
  - A key distribution center (session keys)
  - A certification authority (public keys)
- Identity-based cryptography: your ID is your public key. A trusted authority derives users' private keys (and thus knows *all* private keys!)
- Peer authentication via a *web of trust* that establish the authenticity of the *binding* between a public key and its owner (Phil Zimmerman, 1992, used in PGP secure e-mail)

The vast majority of key distribution systems involve a **trusted authority** to ensure authenticity of keys.