Computer Security
Lecture 14

# Cryptography III:

# Hash Functions & Asymmetric Ciphers

David Aspinall

School of Informatics,

University of Edinburgh.

24th February 2004

http://www.informatics.ed.ac.uk/teaching/modules/cs

# Contents

- Hash functions, MACs
- MD5, SHA-1
- Asymmetric algorithms
- RSA
- Diffie-Hellman key-exchange
- ElGamal, DSA

*Cipher design and cipher breaking were once arts secret themselves. In the last few decades, public science has gained ground. (We think.)*

# Integrity check functions

- Recall that **MDCs** (*modification detection codes*) are either of two varieties of hash function:
  - OWHF: one-way and 2nd pre-image (weak collision) resitance;
  - CRHF: 2nd pre-image resitance and (strong) collision resistance.
- To satisfy (strong) collision resistance, a hash function has to be large enough to withstand a **birthday attack** (or *square root attack*). Drawing random elements with replacement from a set of $n$ elements, a repeated element is likely to be found after $O(\sqrt{n})$ selections.
- Mallory has two contracts, one for €1000, the other €100,000, to be signed with a 64-bit hash. He makes $2^{32}$ minor variations in each (changing spaces or control characters), and finds a pair with the same hash. He can later claim second document was signed, not first.
- An $n$-bit unkeyed hash function has **ideal security** if producing a pre-image or 2nd-pre-image each requires $O(2^n)$ operations, and producing a collision requires $O(2^{n/2})$ operations.

# From one-way functions to MDCs

- **Multiplication of large primes** is a OWF; for appropriate choices of $p$ and $q$, $f(p,q) = pq$ is a one-way function since *integer factorization* is difficult. Not feasible to turn into an MD function, though (**Ex**: why?)
- **Exponentiation in finite fields** (see later) is a OWF; for appropriate primes $p$ and numbers $\alpha$, $f(x) = \alpha^x \bmod p$ is a one-way function, since the *discrete logarithm problem* is difficult. (However, it's easy for some values such as 1, -1). Main problem with turning this into a realistic MD function is that it's too slow to calculate.
- We can construct a **OWF from a block cipher** such as DES, which is treated essentially as a random function:

$$h(x) = E_k(x) \oplus x$$

for fixed key $k$. This *can* be turned into a MD function, by iteration. . .

# Building up hash functions

- An **iterated hash function** is constructed using a *compression function* $f$ which converts a $t+n$-bit input into an $n$-bit output. The input $x$ is split into blocks $x_1 x_2, \ldots x_k$ of size $t+n$, typically by appending padding bits and a *length block* indicating the original length.

$$H_0 = IV \qquad H_i = f(H_{i-1}, x_i),\ \ 1 \le i \le k \qquad h(x) = g(H_k).$$

  IV: an initialization vector; $g$: an output transformation (often identity).

- Fact (**Merkle's meta-method**): any collision-resistant compression function $f$ can be extended to a collision-resistant hash function by the above construction, by padding the last block with 0s, and adding a final extra block $x_k$ which holds right-justified binary representation of $\mathit{length}(x)$ (this padding technique is called **MD strengthening**). Set $IV = 0^n$, $g = \mathit{id}$, and compute $H_i = f(H_{i-1}, x_i)$.

# Outline of MD5

- An improved version of MD4. Both designed by Ron Rivest. Text processed in 512-bit blocks, as 16 32-bit sub-blocks. Output is four 32-bit blocks, giving a 128-bit hash. Message is padded with a 1 and then 0s to 64 bits short of 512*$n$, then a 64-bit length representation.
- Main loop has four rounds, chaining 4 variables $a, b, c, d$. Each round uses a different operation (with a similar structure) 16 times, which computes a new value of one of the four variables using a non-linear function of the other three, chosen to preserve randomness properties of the input. For example, the first round uses the operation:

$$
\begin{aligned}
a &= (F(b,c,d) + x_i + t_j) <<< s \\
F(b,c,d) &= (b \wedge c) \vee (\neg b \wedge d)
\end{aligned}
$$

where $<<< s$ is left-circular shift of $s$ bits, $x_i$ is the $i$th sub-block of the message. Constants $t_j$ are the integer part of $2^{32} * \text{abs}(\sin(i+1))$ where $0 \leq i \leq 63$ is in radians (for the 4 * 16 steps).

# SHA-1

- Secure Hash Algorithm (rev 1) is a NIST standard [FIPS 180] also based on MD4. Five 32-bit blocks are chained; output is 160 bits. Message blocks 512 bits. Padding like MD5. Words are stored in big-endian.
- Main loop has four rounds of 20 operations, chaining 5 variables $a, b, c, d, e, f$. Five IVs and four constants are used:

$$\begin{aligned} A &= \texttt{0x67452301} \\ B &= \texttt{0xEFCDAB89} \\ C &= \texttt{0x98BADCFE} \\ D &= \texttt{0x10325476} \\ E &= \texttt{0xC3D2E1F0} \end{aligned} \qquad \begin{aligned} K_0 &= \texttt{0x5A827999} \\ K_1 &= \texttt{0x6ED9EBA1} \\ K_2 &= \texttt{0x8F1BBCDC} \\ K_3 &= \texttt{0xCA62C1D6} \end{aligned}$$

- The message block undergoes an *expansion transformation* from 16*32-bit words $x_i$ to 80*32-bit words, $w_i$ by:

$$\begin{aligned} w_i &= x_i, & \text{for } 0 \leq i \leq 15. \\ w_i &= (w_{i-3} \oplus w_{i-8} \oplus w_{i-14} \oplus w_{i-16}) <<< 1, & \text{for } 16 \leq i \leq 79. \end{aligned}$$

- Each operation uses a non-linear function of three of the 5 variables:

$$
\begin{aligned}
F_0(X,Y,Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\
F_1(X,Y,Z) &= X \oplus Y \oplus Z \\
F_2(X,Y,Z) &= (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) \\
F_3(X,Y,Z) &= X \oplus Y \oplus Z
\end{aligned}
$$

- Compression function executes this loop, where $j = i/20$:

$$
\begin{aligned}
&\textbf{for}(\ i = 0;\ i < 80;\ i{+}{+}\ ) \qquad \{ \\
&\qquad tmp = (a <<< 5) + F_j(b,c,d) + e + w_i + K_j; \\
&\qquad e = d; \\
&\qquad c = b <<< 30; \\
&\qquad b = a; \\
&\qquad a = tmp; \\
&\}
\end{aligned}
$$

- Finally the variables $a, b, c, d, e$ are added to the previous intermediate value (all addition is modulo $2^{32}$). **Exercise:** implement SHA-1 in your favourite language following this. A 3-letter test: `abc` hashes to `84983e441c3bd26ebaae4aa1f95129e5e54670f1`.

# Block ciphers from hash functions

- We can also construct ciphers from hash functions. To use a hash-function as block cipher in CFB mode: concatentate plaintext block with key and previous ciphertext block ($||$ means concatenation):

$$C_i = P_i \oplus h(K||C_{i-1}) \qquad P_i = C_i \oplus h(K||C_{i-1})$$

  A similar construction using $h$ in OFB mode is possible.

- The **Message Digest Cipher** construction is similar. It uses a function which converts $t$ bits to $n$ bits and is normally seeded with an $n$-bit IV. A $t$-bit key is used as the unchanging input:

$$C_i = P_i \oplus h(C_{i-1}, K) \qquad P_i = C_i \oplus h(C_{i-1}, K)$$

  E.g., SHA would be used with a 512-bit key and 160-bit block size.

- The **Luby-Rackoff** construction uses three hash functions to make a provably secure 3-round Feistel cipher.

- In general one should be wary of these constructions...

# The Random Oracle Model [BR93]

- A strategy for provable security. The technique is to "factor out" crypto primitives, and consider them as being perfectly random:
  - — A *public* oracle $\mathcal{R}$ maps inputs into random (possibly unbounded) output. Same input produces same output.
  - — $\mathcal{R}$ models a hash f'n, encryption f'n, or random number generator.
  - — Then make assumptions on limits to access of the oracle (e.g., a polynomially-bounded adversary), and prove results about a particular usage (e.g., secure against feasible chosen-text attacks).
  - — In the real implementation, $\mathcal{R}$ is replaced by algorithms (e.g. based on DES, MD5, etc). Hope is that the result is still somehow pertinent (an achievable best case) for this setting.
- Used to justify practical constructions in modern cryptography, e.g., RSA-style signature scheme, constructions similar to previous slide.
- But: step of "realizing" $\mathcal{R}$ is risky; ROM hypothesis is shaky.

# Keyed hash functions (MACs)

- Recall that a MAC is a family of hash functions $\{h_k \mid k \in \mathcal{K}\}$ parameterised by secret keys $k \in \mathcal{K}$. Each function $h_k$ must satisfy a particular security requirement (which implies *non recovery* for $k$):
  - *MAC resistance*. For any fixed secret value of $k$, given a set of pairs $(x_i, h_k(x_i))$, it is computationally infeasible to compute $h_k(x)$ for any new input $x$ (including colliding $x$ st $\exists i.\, h_k(x) = h_k(x_i)$).
- Common MAC algorithm: a block-cipher in CBC mode.
- A MAC algorithm can be derived from an MDC algorithm using the **hashed MAC** (HMAC) construction. Given an MDC algorithm $h$, for any given key $k$ and message $x$ compute

$$HMAC_k(x) = h(k \,||\, p_1 || h(k \,||\, p_2 \,|| x)\,)$$

where $p_1$ and $p_2$ are padding which extend $k$ to a full block length of the compression function used in $h$. More obvious simpler constructions than this (e.g., $h(k||x)$, $h(x||k)$ ) are subject to various attacks.

# Hash functions compared

- SHA and MD5 both improved on MD4 by adding an extra round and increasing the *avalanche effect*: how quickly the effect of the input bits spreads in the output. SHA also adds the expand transformation to MD4, so any two different 16-word messages differ give two 80-word values which differ in many bit positions.
- There has been some cryptanalysis of MD5, and collisions have been found for the MD5 compression function, although not for the full hash function itself.
- Someone using a birthday attack on MD5 will have to hash $2^{64}$ random documents to find two that hash to the same value. This is too small a number for long-term security, so 160-bits or greater should be used for long-lived signatures.

# Prime number reminders

- A natural number $p \geq 2$ is *prime* if 1 and $p$ are its only positive divisors. Two integers $a$ and $b$ are *relatively prime* if $\gcd(a, b) = 1$.
- For $x \geq 17$, then $\phi(x)$, the number of primes less than or equal to $x$, is approximated by:

$$\frac{x}{\ln x} < \phi(x) < 1.25506 \frac{x}{\ln x}$$

- Fundamental theorem of arithmetic: every natural $n \geq 2$ has a unique factorization as a product of prime powers: $p_1^{e_1} \cdots p_n^{e_n}$ for distinct primes $p_i$ and postive $e_i$.
- The *Euler totient function* $\phi(n)$ is the number of elements of $\{1, \ldots, n\}$ which are relatively prime to $n$. For prime $n$, $\phi(n) = n - 1$.
- An integer $n$ is said to be $B$-smooth wrt a positive bound $B$, if all its prime factors are $\leq B$. There are efficient algorithms for computing any prime factors $p$ of a compositive integer $n$ for which $p - 1$ is smooth.

## $\mathbf{Z}_n$, the integers modulo $n$

- Let $n$ be a positive integer. Then $\mathbf{Z}_n = \{0, \ldots, n-1\}$, the set of integers modulo $n$ (more properly, the *equivalence classes* $[x]_n$ modulo $n$).
- Let $a \in \mathbf{Z}_n$. The *multiplicative inverse* of $a$ modulo $n$ is the unique $x \in \mathbf{Z}_n$ such that $ax \equiv 1 \pmod{n}$. Fact: $a$ exists iff $\gcd(a, n) = 1$.
- The *multiplicative group* $\mathbf{Z}_n^* = \{a \in \mathbf{Z}_n \mid \gcd(a, n) = 1\}$. Fact: $\mathbf{Z}_n^*$ is closed under multiplication, and $|\mathbf{Z}_n^*| = \phi(n)$.
- Euler's theorem: if $a \in \mathbf{Z}_n^*$, then $a^{\phi(n)} \equiv 1 \pmod{n}$. If $n$ is a product of distinct primes, and if $r \equiv s \pmod{\phi(n)}$, then $a^r \equiv a^s \pmod{n}$. Fermat's theorem: if $p$ prime, $\gcd(a, p) = 1$, then $a^{p-1} \equiv 1 \pmod{n}$.
- Let $a \in Z_n^*$. The *order* of $a$ is the least $t > 0$ st $a^t \equiv 1 \pmod{n}$. If an element $\alpha \in \mathbf{Z}_n^*$ has order $\phi(n)$, then $\mathbf{Z}_n^*$ is *cyclic* and $\alpha$ is a *generator* (*primitive root*) of $\mathbf{Z}_n^*$. Fact: $\mathbf{Z}_n^*$ is cyclic iff $n = 2, 4, p^k, 2p^k$ for prime $p$.
- There is an efficient algorithm for computing discrete logs in $\mathbf{Z}_p^*$ if $p - 1$ has smooth factors.

# Cryptographic Reference Problems

**FACTORING** Integer factorization. Given positive $n$, find its prime factorization, i.e., distint $p_i$ such that $n = p_1^{e_1} \cdots p_n^{e_n}$ for some $e_i \geq 1$.

**RSAP** RSA inversion. Given $n$ such that $n = pq$ for some odd primes $p \neq q$, and $e$ such that $\gcd(e, (p-1), (q-1)) = 1$, and $c$, find $m$ such that $m^e \equiv c \pmod{n}$.

**DLP** Discrete logarithm problem. Given prime $p$, a generator $\alpha$ of $\mathbf{Z}_p^*$, and an element $\beta \in \mathbf{Z}_p^*$, find the integer $x$, with $0 \leq x \leq p-2$ such that $\alpha^x \equiv \beta \pmod{p}$.

**DHP** Diffie-Hellman problem. Given prime $p$, a generator $\alpha$ of $\mathbf{Z}_p^*$, and elements $\alpha^a \bmod p$ and $\alpha^b \bmod p$, find $\alpha^{ab} \bmod p$.

Relationships: RSAP$\leq_P$FACTORING, DHP$\leq_P$DLP, where $\leq_P$ means there is a polytime reduction from first prob to second (first no harder than second).

# RSA

- A key-pair is based on product of two large, distinct, random secret primes, $n = pq$ with $p$ and $q$ roughly the same size, together with a random integer $e$ with $1 < e < \phi$ and $\gcd(e, \phi) = 1$, where $\phi = \phi(n) = (p-1)(q-1)$. Public key is $(n, e)$ and $n$ is the *modulus*.
- Private key is $d$, the unique integer such that $ed \equiv 1 \pmod{\phi}$.
- Message and cipher space $\mathcal{M} = \mathcal{C} = \{0, \ldots, n-1\}$. Encryption is exponentiation with public key $e$, decryption is exponentiation with private key $d$.

$$\begin{aligned} E_{(n,e)}(m) &= m^e \bmod n \\ D_d(c) &= c^d \bmod n \end{aligned}$$

- Decryption works because, for some $k$, $ed = 1 + k\phi$ and

$$(m^e)^d \equiv m^{ed} \equiv m^{1+k\phi} \equiv m m^{k\phi} \equiv m \pmod{n}$$

using Fermat's theorem. (**Exercise:** fill in the details of the proof).

# RSA notes

- RSA is an example of a **reversible** public-key encryption scheme. It's reversible because $e$ and $d$ are symmetric in the definition. RSA digital signatures are defined using this fact (see Cryptography I lecture).
- RSA is often used with randomization (e.g., **salting** with a random appendix) to prevent chosen-plaintext and other attacks.
- Most popular and cryptanalyzed public-key algorithm. Largest modulus factored in RSA challlenge was 155 bits in 1999, which took 8000 MIPS years on a variety of machines. This has been repeated since with less effort, so a 512-bit RSA modulus is not nowadays regarded as secure enough. It's believed that a 1024-bit number will need an advance in mathematics, however.
- To win $10,000, factor RSA 576: 188198812920607963838697239461650439807163563379417382700763356422988859715234665485319060606504743045317388011303396716199692321205734031879550656996221305168759307650257059.

# Diffie-Hellman key agreement

- Diffie-Hellman key agreement allows two principles to agree on a key without authentication. Initial setup: choose and publish a large "secure" prime $p$ and generator $\alpha$ of $\mathbf{Z}_p^*$.

$$\text{Message 1.} \quad A \to B: \quad \alpha^x \bmod p$$

$$\text{Message 2.} \quad B \to A: \quad \alpha^y \bmod p$$

— $A$ chooses random secret $x$, $1 \leq x \leq p-2$, and sends message 1.

— $B$ chooses random secret $y$, $1 \leq y \leq p-2$, and sends message 2.

— $B$ receives $\alpha^x$ and computes shared key as $K = (\alpha^x)^y \bmod p$.

— $A$ receives $\alpha^y$ and computes shared key as $K = (\alpha^y)^x \bmod p$.

- Security rests on intractability of DHP for $p$ and $\alpha$. Protocol is safe against passive adversaries, but not active ones.
  **Exercise:** try some artificial examples with $p = 11$, $\alpha = 2$. Show a MIM attack against the protocol.

# ElGamal encryption

- A key-pair is based on a large random prime $p$ and generator $\alpha$ of $\mathbf{Z}_p^*$, and a random integer $d$. Public key: $(p, \alpha, \alpha^d \bmod p)$, private key: $d$.
- The message space $\mathcal{M} = \{0, \ldots, p-1\}$, and the encryption operation is given by selecting a random integer $r$ and computing a pair:

$$E_{(p,\alpha,\alpha^d)}(m) = (\gamma, \delta) \qquad \text{where} \quad \begin{aligned} \gamma &= \alpha^r \bmod p \\ \delta &= m \cdot (\alpha^d)^r \bmod p. \end{aligned}$$

- Decryption takes an element of ciphertext $C = \mathcal{M} \times \mathcal{M}$, and computes:

$$D_d(\gamma, \delta) = \gamma^{-d} \cdot \delta \bmod p \qquad \text{where } \gamma^{-d} = \gamma^{p-1-d} \bmod p.$$

- Decryption works because $\gamma^{-d} = \alpha^{-dr}$, so

$$D_d(\gamma, \delta) \equiv \alpha^{-dr}\, m\, \alpha^{dr} \equiv m \pmod p.$$

- This is just like using Diffie-Hellman to exchange a session key $\alpha^{dr}$ and then encrypting $m$ by multiplying it with the session key.

# ElGamal signatures

- Same setup as encryption: $p$ is an appropriate prime, $\alpha$ a generator of $\mathbf{Z}_p^*$, and $d$ a random integer with $1 \le d \le p-2$, which is the private signing key. The corresponding public verification key is $(p, \alpha, \alpha^d \bmod p)$.

- To sign a message $m$, $0 \le m \le p$, the user picks a random number $r$ with $1 \le r \le p-2$ and $\gcd(r, p-1) = 1$, and computes:

$$\mathbf{S}_d(m) = (\gamma, \delta) \qquad \text{where} \quad \gamma = \alpha^r \bmod p$$
$$d \cdot \gamma + r \cdot \delta \equiv m \pmod{p-1}.$$

- The verification function checks that $0 < \gamma < p$, and an equation:

$$\mathbf{V}_{(p,\alpha,\alpha^d)}(m, (\gamma, \delta)) = \begin{cases} \text{true} & \text{if } (\alpha^d)^\gamma \cdot \gamma^\delta \equiv \alpha^m \pmod{p}, \\ \text{false} & \text{otherwise.} \end{cases}$$

- Verification works because for a correct signature,

$$(\alpha^d)^\gamma \cdot \gamma^\delta \equiv \alpha^{d\gamma + r\delta} \equiv \alpha^m \pmod{p}.$$

# From ElGamal to DSA

- The Digital Signature Algorithm is part of the *Digitial Signature Standard* NIST standard [FIPS 186] which is based on the ElGamal signature scheme, but with improved efficiency. It was the first digital signature scheme to be recognized by any government.
- Based on two primes: $p$, which is 512–1024 bits long, and $q$, which is a 160-bit prime factor of $p - 1$. A signature signs a SHA-1 hash value of a message. (In fact, ElGamal signing ought also to be used with a hash function, to prevent *existential forgery* attacks as outlined in Cryptography I). For more details of DSA, see e.g., [Gol99].
- **Security** of both ElGamal and DSA schemes relies on the intractibility of the DLP.
- Comparison with RSA signature scheme: key generation is faster; signature generation is about the same; DSA verification is slower. Verification is the most common operation in general.

# Notes about ElGamal

- ElGamal is an example of a **randomized** encryption scheme, so no need to add salt. Security relies in intractability of DHP. Choosing different $r$ for different messages is critical. **Exercise:** why?
- Efficiency: ciphertext is twice as long as the plaintext. Encryption requires two modular exponentiations, which can be sped up by picking the random $r$ with some additional structure (with care).
- The prime $p$ and generator $\alpha$ can be fixed for the system, reducing the size of public keys. Then exponentiation can be speeded up by precomputation; however, so can the best-known algorithm for calculating discrete logarithms, so a larger modulus would be warranted.
- The **security** of ElGamal encryption and signing is based on the intractability of the DHP for $p$. Several other conditions are required, for example, to avoid *weak generators* and feasible attack by the *Pohlig-Hellman* algorithm for computing discrete logarithms.

# Current hash and PK algorithms

**Hash functions**

- **MD5**, **SHA-1** already described.
- **RIPEMD-160** is based on MD4, developed after analysing **RIPEMD**, MD4, and MD5. Uses two side-by-side runs of compression function, combining two 160-bit blocks. Security similar to SHA-1.
- **SHA256** and **SHA512** are NIST proposals for longer hash functions, to provide better than $2^{80}$ work factor.

**Public key schemes**

- **RSA**, **ElGamal** already described.
- **Elliptic curve** schemes. Use ElGamal techniques. Shorter keys.
- **Probablistic** schemes, which achieve **provable security**.

**Digital signatures**

- **RSA**, **ElGamal**, **DSA** already described.
- Several variants of ElGamal, including schemes with *message-recovery*.
- Schemes for **one-time** signatures (e.g., Rabin, Merkle) require a fresh public key for each use. Typically much more efficient than RSA/ElGamal methods.

# References

See Chapter 12 of Gollmann [Gol99] for much of what's covered here. For more details, good sources are [Sma03, Sti02, MOV97, Sch96]. For the latest research in Cryptography, see e.g., the *International Association for Cryptologic Research*, at http://www.iacr.org.


[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[Gol99] Dieter Gollmann. *Computer Security*. John Wiley & Sons, 1999.

[MOV97] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone, editors. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and Its Applications. CRC Press, 1997. Online version at http://cacr.math.uwaterloo.ca/hac.

[Sch96] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, second edition, 1996.

[Sma03] Nigel Smart. *Cryptography: An Introduction*. McGraw-Hill, 2003.

[Sti02] Douglas R Stinson. *Cryptography Theory and Practice*. CRC Press, second edition edition, 2002.