



Applied Cryptography

CO 487



Alfred Menezes

Preface

Disclaimer Much of the information on this set of notes is transcribed directly/indirectly from the lectures of CO 487 during Winter 2021 as well as other related resources. I do not make any warranties about the completeness, reliability and accuracy of this set of notes. Use at your own risk.

For any questions, send me an email via <https://notes.sibeliusp.com/contact>.

You can find my notes for other courses on <https://notes.sibeliusp.com/>.

Sibelius Peng

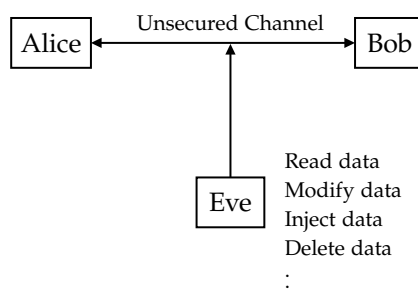
Contents

Preface	1
1 Introduction	4
1.1 Secure Web Transactions	5
1.2 The TLS Protocol	5
1.3 Cryptography in Context	6
2 Symmetric-Key Cryptography	8
2.1 Basic concepts	8
2.1.1 The Simple Substitution Cipher	9
2.1.2 Polyalphabetic Ciphers	12
2.2 The One-Time Pad	12
2.3 Stream Ciphers	13
2.4 The RC4 Stream Cipher	14
2.4.1 Wired Equivalent Privacy	15
2.4.2 Fluhrer-Mantin-Shamir Attack	17
2.5 ChaCha20 Stream Cipher	19
2.5.1 ChaCha20 Quarter Round Function	20
2.6 Block Ciphers	21
2.6.1 Brief History of Block Ciphers	21
2.6.2 Some Desirable Properties of Block Ciphers	22
2.6.3 The Data Encryption Standard (DES)	22
2.6.4 Double-DES	23
2.6.5 Triple-DES	25
2.6.6 The Advanced Encryption Standard (AES)	25
2.6.7 Substitution-Permutation Networks	26
2.6.8 AES	27
2.6.9 Block Cipher Modes of Operation	31
3 Hash Functions	33
3.1 Generic Attacks	37
3.2 Iterated Hash Functions (Merkle Meta-Method)	40
3.2.1 Collision Resistance of Iterated Hash Functions	40
3.3 Provable Security	40
3.4 MDx-Family of Hash Functions	41
3.4.1 MD5 Hash Function	41
3.5 SHA-1	42
3.6 SHA-2 Family	42
3.7 Description of SHA-256	43

3.7.1	SHA-256 Constants	43
3.8	SHA-3	44
4	Message authentication code schemes	45

Introduction

Cryptography is about securing communications in the presence of *malicious* adversaries.



Note that even if we call him “Eve”, he can do more than eavesdropping... The adversary is malicious, powerful and unpredictable.

Fundamental Goals of Cryptography

1. Confidentiality: Keeping data secret from all but those authorized to see it.
2. Data integrity: Ensuring data has not been altered by unauthorized means.
3. Data origin authentication: Corroborating the source of data.
4. Non-repudiation: Preventing an entity from denying previous commitments or actions.

Some examples of unsecured channel:

- Secure Browsing: The Internet
- Online Shopping: The Internet
- Automatic Software Upgrades: The Internet
- Cell Phone Service: Wireless
- Wi-Fi: Wireless
- Bluetooth: Wireless
- Messaging: Wired/Wireless

Communicating Parties

Alice and Bob are two communicating devices.

Alice	Bob	Communication channel
person	person	telephone cable
person	person	cellular network
person	web site	internet
iPhone	wireless	router wireless
iPhone	headphones	wireless
iPhone	service provider	cellular network
your car's brakes	another car	wireless
smart card	bank machine	financial network
smart meter	energy provider	wireless
military commander	satellite	space

1.1 Secure Web Transactions

Transport Layer Security (TLS): The cryptographic protocol used by web browsers for secure web transactions for secure access to amazon, gmail, hotmail, facebook etc.

TLS is used to assure an individual user (called a client) of the authenticity of the web site (called the server) he or she is visiting, and to establish a secure communications channel for the remainder of the session.

Symmetric-key cryptography: The client and server a priori share some secret information k , called a key.

They can subsequently engage in secure communications by encrypting their messages with AES and authenticating the resulting ciphertexts with HMAC.

How do they establish the shared secret key k ?

Public-key cryptography: Communicating parties a priori share some authenticated (but non-secret) information.

To establish a secret key, the client selects the secret session key k , and encrypts it with the server's RSA public key. Then only the server can decrypt the resulting ciphertext with its RSA private key to recover k .

How does the client obtain an authentic copy of the server's RSA public key?

Signature scheme: The server's RSA public key is signed by a Certifying Authority using the RSA signature scheme.

The client can verify the signature using the Certifying Authority's RSA public verification key which is embedded in its browser. In this way, the client obtains an authentic copy of the server's RSA public key.

1.2 The TLS Protocol

1. When a client first visits a secured web page, the server transmits its certificate to the client.
 - The certificate contains the server's identifying information (e.g., web site name and URL) and RSA public key, and the RSA signature of a certifying authority.

- The certifying authority (e.g., Verisign) is trusted to carefully verify the server's identity before issuing the certificate.
2. Upon receipt of the certificate, the client verifies the signature using the certifying authority's public key, which is embedded in the browser. A successful verification confirms the authenticity of the server and of its RSA public key.
 3. The client selects a random session key k , encrypts it with the server's RSA public key, and transmits the resulting ciphertext to the server.
 4. The server decrypts the ciphertext to obtain the session key, which is then used with symmetric-key schemes to encrypt (e.g. with AES) and authenticate (e.g. with HMAC) all sensitive data exchanged for the remainder of the session.
 5. The establishment of a secure link is indicated by a closed padlock in the browser. Clicking on this icon reveals the server's certificate and information about the certifying authority.

TLS is one of the most successful security technologies ever deployed. But is TLS really secure?

There are many potential security vulnerabilities:

1. The crypto is weak (e.g., AES, HMAC, RSA).
2. Quantum attacks on the underlying cryptography.
3. Weak random number generation.
4. Issuance of fraudulent certificates
 - In 2001, Verisign erroneously issued two Class 3 code-signing certificates to a person masquerading as a Microsoft representative.
 - Mistake due to human error.
5. Software bugs (both inadvertent and malicious).
6. Phishing attacks.
7. TLS only protects data during transit. It does not protect your data when it is collected at the server.

Many servers store large amounts of credit card data and other personal information.

8. The National Security Agency (NSA)

1.3 Cryptography in Context

Cybersecurity is comprised of the concepts, technical measures, and administrative measures used to protect networks, computers, programs and data from deliberate or inadvertent unauthorized access, disclosure, manipulation, loss or use. Also known as information security. Cybersecurity includes the study of computer security, network security and software security.

Note that Cryptography \neq Cybersecurity.

- Cryptography provides some mathematical tools that can assist with the provision of cybersecurity services. It is a small, albeit an indispensable, part of a complete security solution.
- Security is a chain
 - Weak links become targets; one flaw is all it takes.

- Cryptography is usually not the weakest link. However, when the crypto fails the damage can be catastrophic.

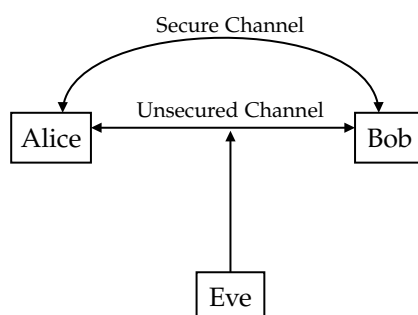
Symmetric-Key Cryptography

2.1 Basic concepts

symmetric-key encryption scheme

A **symmetric-key encryption scheme (SKES)** consists of:

- M - the plaintext space,
- C - the ciphertext space,
- K - the key space,
- a family of encryption functions: $E_k : M \rightarrow C, \forall k \in K$,
- a family of decryption functions: $D_k : C \rightarrow M, \forall k \in K$, such that $D_k(E_k(m)) = m$ for all $m \in M, k \in K$.



1. Alice and Bob agree on a secret key $k \in K$ by communicating over the secure channel.
2. Alice computes $c = E_k(m)$ and sends the ciphertext c to Bob over the unsecured channel.
3. Bob retrieves the plaintext by computing $m = D_k(c)$.

Some examples:

- WWII: Enigma Machine, Alan Turing,
https://en.wikipedia.org/wiki/Cryptanalysis_of_the_Enigma
- WWII: Lorenz Machine, Bill Tutte, Colossus

<http://tinyurl.com/COBillTutte>, <http://billtuttememorial.org.uk/>

2.1.1 The Simple Substitution Cipher

- M = all English msgs.
- C = all encrypted msgs.
- K = all permutations of the English alphabet.
- $E_k(m)$: Apply permutation k to m , one letter at a time.
- $D_k(c)$: Apply inverse permutations k^{-1} to c , one letter at a time.

Example:

$k =$

<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>m</i>	<i>n</i>	<i>o</i>	<i>p</i>	<i>q</i>	<i>r</i>	<i>s</i>	<i>t</i>	<i>u</i>	<i>v</i>	<i>w</i>	<i>x</i>	<i>y</i>	<i>z</i>
<i>D</i>	<i>N</i>	<i>X</i>	<i>E</i>	<i>S</i>	<i>K</i>	<i>O</i>	<i>J</i>	<i>T</i>	<i>A</i>	<i>F</i>	<i>P</i>	<i>Y</i>	<i>I</i>	<i>Q</i>	<i>U</i>	<i>B</i>	<i>R</i>	<i>Z</i>	<i>G</i>	<i>V</i>	<i>C</i>	<i>H</i>	<i>M</i>	<i>W</i>	<i>L</i>

Encryption: $m = \text{the big dog}$, $c = E_k(\text{the big dog}) = \text{GJS NTO EQO}$.

Decryption: $c = \text{GJS NTO EQO}$, $m = E_k^{-1}(\text{GJS NTO EQO}) = \text{the big dog}$.

Is the simple substitution cipher a secure SKES? but wait, what does it mean for a SKES to be secure? We need a **security definition**.

1. What are the computational powers of the adversary?
2. How does the adversary interact with the two communicating parties?
3. What is the adversary's goal?

Security model: Defines the computational abilities of the adversary, and how she interacts with the communicating parties.

Basic assumption: The adversary knows everything about the SKES, except the particular key k chosen by Alice and Bob. (Avoid security by obscurity!!) Security should only depend on the secrecy of the secret keying material, not on the secrecy of the algorithms that describe the cryptographic scheme.

Let's now consider the computational power of the adversary.

- **Information-theoretic security:** Eve has infinite computational resources.
- **Complexity-theoretic security:** Eve is a 'polynomial-time Turing machine'.
- **Computational security:** Eve has 36,768 Intel E5-2683 V4 cores running at 2.1 GHz at her disposal. See: **Graham** in the basement of MC

We say: Eve is "*computationally bounded*".

In this course, we will be concerned with computational security.

Next consider how the adversary can interact with Alice and Bob.

- Passive attacks:
 - **Ciphertext-only attack:** The adversary knows some ciphertext (that was generated by Alice or Bob).
 - **Known-plaintext attack:** The adversary also knows some plaintext and the corresponding ciphertext. (stronger because more info than before)
- Active attacks:

- **Chosen-plaintext attack:** The adversary can also choose some plaintext and obtains the corresponding ciphertext. This is at least as strong as known-plaintext attack because the attacker gets to choose the plain text for which it is given the corresponding ciphertext.
- Other attacks (not considered in this course):
 - Clandestine attacks: bribery, blackmail, etc.
 - Side-channel attacks: monitor the encryption and decryption equipment (timing attacks, power analysis attacks, electromagnetic-radiation analysis, etc.)

Finally, we will consider the adversary's goal (from strongest to weakest).

1. Recover the secret key.
2. Systematically recover plaintext from ciphertext (without necessarily learning the secret key).
3. Learn some partial information about the plaintext from the ciphertext (other than its length).

If the adversary can achieve 1 or 2, the SKES is said to be **totally insecure** (or **totally broken**).

If the adversary cannot learn any partial information about the plaintext from the ciphertext (except possibly its length), the SKES is said to be **semantically secure**. So the ciphertext hides all the meaning about the corresponding plaintext.

security of SKES

A symmetric-key encryption scheme is said to be **secure** if it is semantically secure against chosen-plaintext attack by a computationally bounded adversary.

Note that the definitions captures the computational abilities of the adversary, namely some concrete upper bound on its computational resources. It captures how the adversary interacts with Alice and Bob, namely by mounting a chosen-plaintext attack. And finally, it captures the goal of the adversary, namely breaking semantic security. Note also that in the definition the attacker's capabilities are *as strong as possible*, namely a chosen-plaintext attack and its goal is *as weak as possible* namely, breaking semantic security

From the definition, we can also deduce what it means to break an encryption scheme.

1. The adversary is given a challenge ciphertext c (generated by Alice or Bob using their secret key k).
2. During its computation, the adversary can select plaintexts and obtains (from Alice or Bob) the corresponding ciphertexts.
3. After a feasible amount of computation, the adversary obtains some information about the plaintext m corresponding to the challenge ciphertext c (other than the length of m).

If the attacker can win this game specified in these three steps, then we'll say that the encryption scheme is insecure.

Desirable Properties of a SKES

1. Efficient algorithms should be known for computing E_k and D_k (i.e., for encryption and decryption).
2. The secret key should be small (but large enough to render exhaustive key search infeasible).
3. The scheme should be secure.

4. The scheme should be secure even against the designer of the system.

The last point is to ensure that the designer hasn't somehow inserted a backdoor into the encryption scheme whereby the adversary can learn a secret keying material by looking at ciphertext, while this method would be hard to find by other people. This fourth property can sometimes be very difficult to obtain in practice.

Now we can see whether the simple substitution cipher is secure: Totally insecure against a chosen-plaintext attack. This is because the adversary can simply give the plaintext message that's comprised of the English alphabet from a to z to Alice for encryption. Alice returns to the adversary the ciphertext, which it turns out is the secret key itself. And so by simply giving Alice a short plaintext and getting back the corresponding ciphertext the adversary has learnt Alice's secret key.

What about security under a ciphertext-only attack? Is exhaustive key search possible?

- Given sufficient amounts of ciphertext c , decrypt c using each possible key until c decrypts to a plaintext message which "makes sense".
- In principle, 30 characters of ciphertext are sufficient on average to yield a unique plaintext that is a sensible English message. In practice, a few hundred characters are needed.

Exhaustive search:

- Number of keys to try is $26! \approx 2^{88}$
- If the adversary uses 10^6 computers, each capable of trying 10^9 keys per second, then exhaustive key search takes about 10^4 years.
- So, exhaustive key search is infeasible.

In this course,

- 2^{40} operations is considered very easy.
- 2^{56} operations is considered easy.
- 2^{64} operations is considered feasible.
- 2^{80} operations is considered barely feasible.
- 2^{128} operations is considered infeasible.

The Bitcoin network is presently performing hash operations at the rate of $2^{66.4}$ per second (or $2^{91.3}$ per year).

The Landauer limit from thermodynamics suggests that exhaustively trying 2^{128} symmetric keys would require $\gg 3000$ gigawatts of power for one year (which is $\gg 100\%$ of the world's energy production). http://en.wikipedia.org/wiki/Brute-force_attack

security level

A cryptographic scheme is said to have a **security level** of ℓ bits if the fastest known attack on the scheme takes approximately 2^ℓ operations.

As of the year 2021, a security level of 128 bits is desirable in practice.

Of course, simple frequency analysis of ciphertext letters can be used to recover the key. One would simply find the most frequently occurring ciphertext letter; this would most likely correspond to the English letter e which is the most frequently occurring letter in the English alphabet. Hence, the

simple substitution cipher is totally insecure even against a ciphertext-only attack.

2.1.2 Polyalphabetic Ciphers

Basic idea: Use several permutations, so a plaintext letter is encrypted to one of several possible ciphertext letters.

Example: [Vigenère cipher](#)

Secret key is an English word having no repeated letters. E.g., $k = \text{CRYPTO}$.

Example of encryption:

$$\begin{array}{rcccccccccccccccc}
 m & = & t & h & i & s & i & s & a & m & e & s & s & a & g & e \\
 + & k & = & C & R & Y & P & T & O & C & R & Y & P & T & O & C & R \\
 \hline
 c & = & V & Y & G & H & B & G & C & D & C & H & L & O & I & V
 \end{array}$$

Here, $A = 0, B = 1, \dots, Z = 25$; addition of letters is mod 26.

Decryption is subtraction modulo 26: $m = c - k$.

Frequency distribution of ciphertext letters is flatter (than for a simple substitution cipher).

The Vigenère cipher is totally insecure against a chosen-plaintext attack. Not unexpectedly, the Vigenère cipher is also totally insecure against a ciphertext-only attack.

2.2 The One-Time Pad

The one-time pad is a symmetric-key encryption scheme invented by Vernam in 1917 for the telegraph system. The key is a random¹ string of letters. Example of encryption:

$$\begin{array}{rcccccccccccccccc}
 m & = & t & h & i & s & i & s & a & m & e & s & s & a & g & e \\
 + & k & = & Z & F & K & W & O & G & P & S & M & F & J & D & L & G \\
 \hline
 c & = & S & M & S & P & W & Y & P & F & Q & X & C & D & R & K
 \end{array}$$

Note that the key is as long as the plaintext. One major disadvantage of the one-time pad is that the secret key is of the same length as the plaintext. This is especially inconvenient if Alice and Bob are exchanging long messages over a period of time, and so it might be tempting in practice to reuse the secret key. The key should *not* be re-used: If $c_1 = m_1 + k$ and $c_2 = m_2 + k$, then $c_1 - c_2 = m_1 - m_2$. Thus $c_1 - c_2$ depends only on the plaintext (and not on the key) and hence can leak information about the plaintext. In particular, if m_1 is known, then m_2 can be easily computed.

Convention: From now on, unless otherwise stated, messages and keys will be assumed to be binary/bit strings.

⊕

⊕ is bitwise exclusive-or (XOR) i.e., bitwise addition modulo 2. For example:

$$1011001010 \oplus 1001001001 = 0010000011.$$

Note that $x \oplus x = 0$ and $x \oplus y = y \oplus x$. Hence if $x = y \oplus z$, then $x \oplus y = z$.

So, for the one-time pad:

$$\text{Encryption: } c = m \oplus k.$$

¹independently and uniformly at random

Decryption: $m = c \oplus k$.

<https://cryptosmith.com/2008/05/31/stream-reuse/> is an example which reuses the same key...

Security of One-Time Pad

Perfect secrecy: The one-time pad is semantically secure against ciphertext-only attack by an adversary with infinite computational resources. This can be proven formally using concepts from information theory [Shannon 1949].

The bad news: Shannon (1949) proved that if plaintexts are m -bit strings, then any symmetric-key encryption scheme with perfect secrecy must have $|K| \geq 2^m$. So, perfect secrecy (and the one-time pad) is fairly useless in practice. However, all is not lost. One can use one-time pads to inspire the construction of so-called stream ciphers.

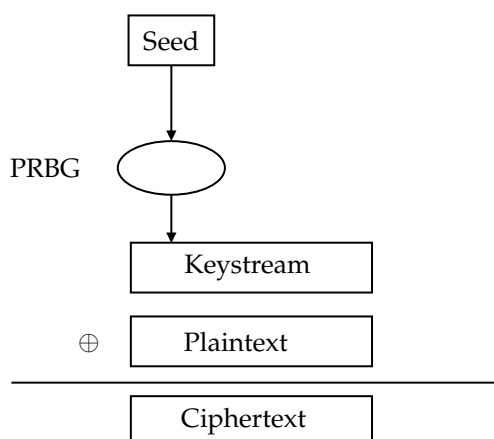
2.3 Stream Ciphers

Basic idea: Instead of using a random key in the one-time pad, use a “pseudorandom” key.

A **pseudorandom bit generator** (PRBG) is a deterministic algorithm that takes as input a (relatively small random) seed, and outputs a longer “pseudorandom” sequence called the keystream.

Using a PRBG for encryption (a stream cipher):

The seed is the secret key shared by Alice and Bob.



The XOR portion of the stream cipher emulates a one-time pad. Note though that we no longer have perfect secrecy because the keystream is no longer purely random but pseudo-random. Thus security depends on the quality of the PRBG.

Security Requirements for the PRBG

The keystream should be “indistinguishable” from a random sequence (the **indistinguishability requirement**).

If an adversary knows a portion c_1 of ciphertext and the corresponding plaintext m_1 , then she can easily find the corresponding portion $k_1 = c_1 \oplus m_1$ of the keystream. Thus, given portions of the keystream, it should be infeasible to learn any information about the rest of the keystream (the **unpredictability requirement**).

Aside: Don’t use UNIX random number generators (rand and srand) for cryptography!

- $X_0 = \text{seed}, X_{i+1} = aX_i + b \pmod n, i \geq 0$.

- a, b and n are fixed and public constants.

And so the UNIX routines `rand` and `srand` do not meet the unpredictability requirement.

One difficulty with using stream ciphers in practice is that we still have the requirement as we did with the one-time pad that keystream should never be reused.

2.4 The RC4 Stream Cipher

Designed by Ron Rivest in 1987.

Until recently, was widely used in commercial products including Adobe Acrobat, Windows password encryption, Oracle secure SQL, TLS, etc.

- *Pros:* Extremely simple; extremely fast; variable key length. No catastrophic weakness has been found since 1987.
- *Cons:* Design criteria are proprietary; not much public scrutiny until the year 2001.

In the past 10 years, many weaknesses have been found in RC4 and so its use has rapidly declined in practice.

RC4 has two components: (i) a *key scheduling algorithm*, and (ii) a *keystream generator*.

In the following, $K[i]$, $\bar{K}[i]$ and $S[i]$ are 8-bit integers (bytes).

Algorithm 1: RC4 Key Scheduling Algorithm

Input: Secret key $K[0], K[1], \dots, K[d-1]$. (Keylength is $8d$ bits.)

Output: 256-long array: $S[0], S[1], \dots, S[255]$.

```

1 for  $i = 0, \dots, 255$  do
2    $S[i] \leftarrow i$ 
3    $\bar{K}[i] \leftarrow K[i \bmod d]$ 
4  $j \leftarrow 0$ 
5 for  $i = 0, \dots, 255$  do
6    $j \leftarrow (\bar{K}[i] + S[i] + j) \bmod 256$ 
7   Swap( $S[i], S[j]$ )
```

Idea: S is a “random-looking” permutation of $\{0, 1, 2, \dots, 255\}$ that is generated from the secret key.

Algorithm 2: RC4 Keystream Generator

Input: 256-long byte array: $S[0], S[1], \dots, S[255]$ produced by the RC4 Key Scheduling Algorithm

Output: Keystream.

```

1  $i \leftarrow 0; j \leftarrow 0$ 
2 while keystream bytes are required do
3    $i \leftarrow (i + 1) \bmod 256$ 
4    $j \leftarrow (S[i] + j) \bmod 256$ 
5   Swap( $S[i], S[j]$ )
6    $t \leftarrow (S[i] + S[j]) \bmod 256$ 
7   Output( $S[t]$ )
```

ENCRYPTION: The keystream bytes are stored with the plaintext bytes to produce ciphertext bytes.

2.4.1 Wired Equivalent Privacy

Wireless Security

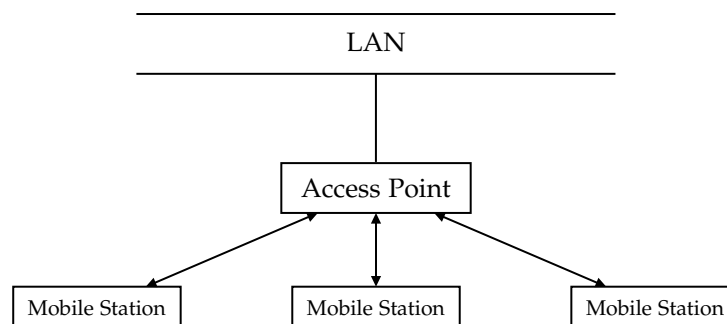
Wireless networks have become prevalent. Popular standards for wireless networks: IEEE 802.11 (longer range, higher speeds, commonly used for wireless LANs) and Bluetooth (short range, low speed).

New security concerns:

- More attack opportunities (no need for physical access).
- Attack from a distance (> 1 km with good antennae).
- No physical evidence of attack.

Case Study: IEEE 802.11 Security

IEEE 802.11 standard for wireless LAN communications includes a protocol called **Wired Equivalent Privacy** (WEP). This standard was ratified in September 1999. This was a very exciting time because this was the first time that wi-fi was available to the consumer market. Multiple amendments: 802.11a (1999), 802.11b (1999), 802.11g (2003), 802.11i (2004), 802.11n (2009)... WEP's goal is (only) to protect link-level data during wireless transmission between mobile stations and access points. A mobile station might be your cell phone or your laptop, and an access point is a wi-fi router somewhere in the building.



Main Security Goals of WEP

1. *Confidentiality*: Prevent casual eavesdropping.
To achieve this, RC4 is used for encryption.
2. *Data Integrity*: Prevent tampering with transmitted messages.
To achieve this, an 'integrity checksum' is used for WEP.
3. *Access Control*: Protect access to a wireless network infrastructure.

This was achieved by discarding all packets that are not properly encrypted using WEP.

Description of WEP Protocol

Mobile stations share a secret key k with access point. Here k is either 40 bits or 104 bits in length. The IEEE standard does not specify how the key is to be distributed. But in practice, one shared key per LAN is common; this key is manually injected into each access point and mobile station; the key is not changed very frequently.

Plaintext messages are divided into packets of some fixed length (e.g., 1500 bytes). These packets were

encrypted using RC4. Since a mobile station or access point might encrypt many packets in a given period of time, WEP had to be careful not to reuse RC4 keystreams. So WEP uses a per-packet 24-bit initialization vector (IV) v to process each packet. WEP does not specify how the IVs are managed. In practice: either a random IV is generated for each packet; or the IV is set to 0 and incremented by 1 for each use.

To send a packet m , an entity does the following:

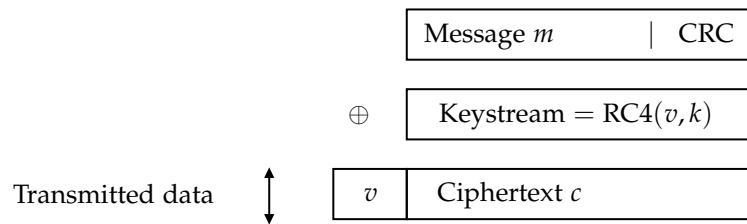
1. Select a 24-bit IV v .
2. Compute a 32-bit checksum: $S = \text{CRC}(m)$.

802.11 specifies that a CRC-32 checksum be used. CRC-32 is linear. That is, for any two messages m_1 and m_2 of the same bitlength,

$$\text{CRC}(m_1 \oplus m_2) = \text{CRC}(m_1) \oplus \text{CRC}(m_2).$$

(The details of CRC-32 are not important to us)

3. Compute $c = (m \parallel S) \oplus \text{RC4}(v \parallel k)$.
 - \parallel (and also a comma) denotes concatenation
 - $(v \parallel k)$ is the key used the RC4 stream cipher.
4. Send (v, c) over the wireless channel.



The receiver of (v, c) does the following:

1. Compute $(m \parallel s) = c \oplus \text{RC4}(v \parallel k)$.
2. Compute $S' = \text{CRC}(m)$; reject the packet if $S' \neq s$.

Are confidentiality, data integrity, and access control achieved? Intuitively the answer is yes. No (Borisov, Goldberg & Wagner; 2001),

Problem 1: IV Collision

Suppose that two packets (v, c) and (v, c') use the same IV v . Let m, m' be the corresponding plaintexts. Then $c \oplus c' = (m \parallel S) \oplus (m' \parallel S')$. Thus, the eavesdropper can compute $m \oplus m'$. If m is known, then m' is immediately available. If m is not known, then one may be able to use the expected distribution of m and m' to discover information about them. (Some contents of network traffic is predictable.)

Since there are only 2^{24} choices for the IV, collisions are guaranteed after enough time - a few days on a busy network (5 Mbps). If IVs are randomly selected, then one can expect a collision after about 2^{12} packets. This is due to the birthday paradox.

Birthday paradox

Suppose that an urn contains n numbered balls. Suppose that balls are drawn from the urn, one at a time, with replacement. The expected number of draws before a ball is selected for a second time (called a collision) is approximately $\sqrt{\pi n/2} \approx \sqrt{n}$.

Collisions are more likely if keys k are long-lived and the same key is used for multiple mobile stations in a network.

Conclusion: WEP does not provide a high degree of confidentiality.

Problem 2: Checksum is Linear

CRC-32 is used to check integrity. This is fine for random errors, but not for deliberate ones.

It is easy to make controlled changes to (encrypted) packets: Suppose (v, c) is an encrypted packet. Let $c = \text{RC4}(v \parallel k) \oplus (m \parallel S)$, where k, m, S are unknown to the adversary. Let $m' = m \oplus \Delta$, where Δ is a bit string. (The 1's in Δ correspond to the bits of m an attacker wishes to change.) Let $c' = c \oplus (\Delta \parallel \text{CRC}(\Delta))$. Then (v, c') is a valid encrypted packet for m' .

$$\begin{aligned}
 c' &= c \oplus (\Delta \parallel \text{CRC}(\Delta)) \\
 &= \text{RC4}(v \parallel k) \oplus (m \parallel S) \oplus (\Delta \parallel \text{CRC}(\Delta)) \\
 &= \text{RC4}(v \parallel k) \oplus (m \oplus \Delta \parallel \text{CRC}(m) \oplus \text{CRC}(\Delta)) \\
 &= \text{RC4}(c \parallel k) \oplus (m' \parallel \text{CRC}(m \oplus \Delta)) \\
 &= \text{RC4}(v \parallel k) \oplus (m' \parallel \text{CRC}(m'))
 \end{aligned}$$

In this way, the adversary has successfully made a controlled change to the plaintext m without actually knowing the plaintext.

Conclusion: WEP does not provide data integrity.

Problem 3: Integrity Function is Unkeyed

Suppose that an attacker learns the plaintext m corresponding to a single encrypted packet (v, c) . Then, the attacker can compute the RC4 keystream $\text{RC4}(v \parallel k) = c \oplus (m \parallel \text{CRC}(m))$. Henceforth, the attacker can compute a valid encrypted packet for *any* plaintext m' of her choice: (v, c') , where $c' = \text{RC4}(v \parallel k) \oplus (m' \parallel \text{CRC}(m'))$.

Conclusion: WEP does not provide access control.

Optional Reading: Sections 1, 2, 3, 4.1, 4.2, 6 of “[Intercepting mobile communications: The insecurity of 802.11](#)”, by N. Borisov, I. Goldberg and D. Wagner.

2.4.2 Fluhrer-Mantin-Shamir Attack

Shortly after these attacks on WEP were disclosed, a *more devastating attack* was discovered. This was due to Fluhrer, Mantin & Shamir, 2001. Shamir is the “S” in RSA.

The attack makes the following three assumptions:

1. The same 104-bit key k is used for a long period of time. [Most products do this.]
2. The IV is incremented for each packet, or a random IV is selected for each packet. [Most products do this.]

3. The first plaintext byte of each packet (i.e. the first byte of each m) is known to the attacker. [Most wireless protocols prepend the plaintext with some header bytes which are non-secret.]

In the attack, a passive adversary who can collect about 5,000,000 encrypted packets can very easily recover k (and thus totally break the system). [Details not covered in this course.]

The attack can be easily mounted in practice: Can buy a \$100 wireless card and hack drivers to capture (encrypted) packets. On a busy wireless network (5Mbps), 5 million packets can be captured in a few hours, and then k can be immediately computed.

Implementation details: A. Stubblefield, J. Ionnidis, A. Rubin, "Using the Fluhrer, Mantin and Shamir attack to break WEP", AT&T Technical Report, August 2001.

If you want to mount the WEP attack today, all you have to do is find a wireless network that still uses WEP and then go to one of these two websites: [Aircrack-ng](#), [WEPCrack](#). The latest enhancement of the Fleur-Mantin-Shamir attack is aircrack ptw. This can break WEP in under 60 seconds (only about 40,000 packets are needed).

WEP was blamed for the 2007 theft of 45 million credit-card numbers from T.J. Maxx. (T.J. Maxx is an American department store chain) A subsequent class action lawsuit settled for \$40,900,000.

See: <http://tinyurl.com/WEP-TJMaxx>

IEEE 802.11 Update

http://en.wikipedia.org/wiki/Wi-Fi_Protected_Access

WPA2 (Wi-Fi Protected Access)

- Implements the new IEEE 802.11i standard (2004).
- Uses the AES block cipher instead of RC4.
- Deployed ubiquitously in Wi-fi networks.
 - But deployments of WEP are still out there :-(
 - See <https://wifile.net/stats>.
- KRACK attack disclosed in October 2017.

WPA3

- Adopted in January 2018.
- Further improvements to WPA2.
- Dragonblood attack disclosed in April 2019.

The Fluhrer-Mantin-Shamir attack exploits known biases in the first few bytes of the keystream. The attack can be defeated by discarding the first few bytes (e.g., 768 bytes) of the keystream. [Details omitted]

As of 2013, approximately 50% of all TLS traffic was secured using RC4. 2013-2021: Because of several new weaknesses discovered, RC4 has been deprecated in applications such as TLS. (As of October 2019, $\approx 11.6\%$ of websites have RC4 enabled, and only $\approx 1.1\%$ actually use it.)

Conclusions: Don't use RC4. Instead use ChaCha20 or AES-CTR (more on these later).

Lessons Learned

1. Details matter: RC4 was considered to be a secure stream cipher. However, it was improperly used in WEP and the result was a highly insecure communications protocol. Do not assume that “obvious” ways of using cryptographic functions are secure.
2. Attacks only get better; they never get worse.
 - (a) Moore’s law (1965): computers get twice as fast every two years.
 - (b) Known attacks are constantly being tweaked and improved.
 - (c) New attacks are constantly being invented.
3. Designing security is hard:
 - Designing cryptographic protocols is complicated and difficult.
 - Clearly state your security objectives.
 - Hire cryptography experts to design your security. Programming or engineering experts are not good enough.
 - Make your protocols available for public scrutiny.
4. There is a big demand in industry for “security engineers”: People who have a deep understanding of applied cryptography and have excellent programming skills.

2.5 ChaCha20 Stream Cipher

Designed by Dan Bernstein in 2008.

The ChaCha20 stream cipher is conceptually simple, word-oriented, and uses only simple arithmetic operations (integer addition modulo 2^{32} , xor, and left rotations). It is extremely fast in software, and does not require any special hardware. To date, no security weaknesses have been found. ChaCha20 is widely deployed in practice, including in TLS.

Notation:

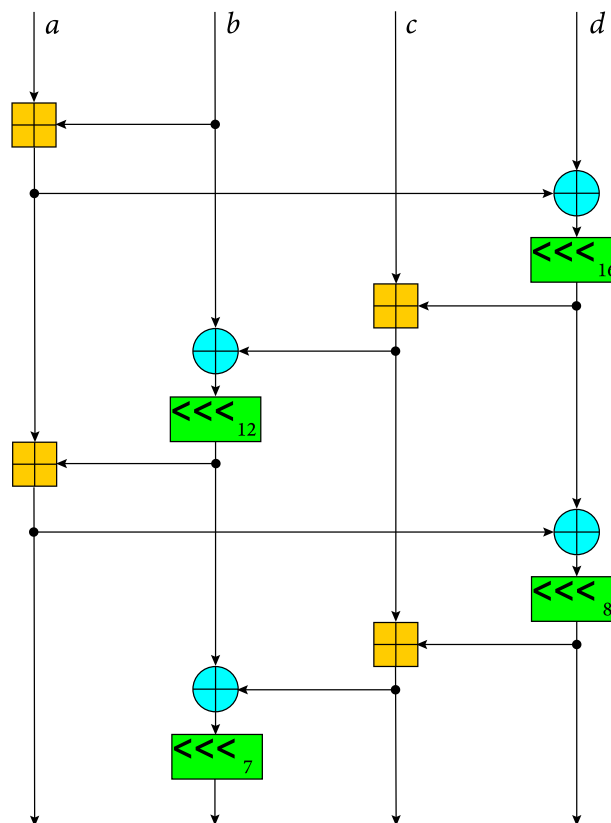
- 256-bit key: $k = (k_1, k_2, \dots, k_8)$
- 96-bit nonce: $n = (n_1, n_2, n_3)$
- 128-bit constant: $f = (f_1, f_2, f_3, f_4)$
- 32-bit counter: c
- The initial state is:

$$\begin{array}{|c|c|c|c|} \hline f_1 & f_2 & f_3 & f_4 \\ \hline k_1 & k_2 & k_3 & k_4 \\ \hline k_5 & k_6 & k_7 & k_8 \\ \hline c & n_1 & n_2 & n_3 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline S_1 & S_2 & S_3 & S_4 \\ \hline S_5 & S_6 & S_7 & S_8 \\ \hline S_9 & S_{10} & S_{11} & S_{12} \\ \hline S_{13} & S_{14} & S_{15} & S_{16} \\ \hline \end{array}$$

A hexadecimal digit is a 4-bit number.

A nonce (or IV) is a non-repeating quantity (either a counter, or a randomly-generated string).

2.5.1 ChaCha20 Quarter Round Function



picture from wiki.

Notation:

- \oplus : XOR
- \boxplus : integer addition modulo 2^{32}
- $\lll t$: left-rotate by t bit positions.

QR: Quarter Round Function

Input Four 32-bit words a, b, c, d

$$\begin{aligned}
 a &\leftarrow a \boxplus b, & d &\leftarrow d \oplus a, & d &\leftarrow d \lll 16 \\
 c &\leftarrow c \boxplus d, & b &\leftarrow b \oplus c, & b &\leftarrow b \lll 12 \\
 a &\leftarrow a \boxplus b, & d &\leftarrow d \oplus a, & d &\leftarrow d \lll 8 \\
 c &\leftarrow c \boxplus d, & b &\leftarrow b \oplus c, & b &\leftarrow b \lll 7
 \end{aligned}$$

Output a, b, c, d

ENCRYPTION: The keystream bytes are xored with the plaintext bytes to produce ciphertext bytes. The nonce is included in the ciphertext.

Algorithm 3: ChaCha20 Keystream Generator

```

1 Select a nonce  $n$  and initialize the counter  $c$ .
2 while keystream bytes are required do
3   Create the initial state  $S$ .
4   Make a copy  $S'$  of  $S$ .
5   Update  $S$  by repeating the following 10 times:

                                    $QR(S_1, S_5, S_9, S_{13})$    $QR(S_2, S_6, S_{10}, S_{14})$ 
                                    $QR(S_3, S_7, S_{11}, S_{15})$    $QR(S_4, S_8, S_{12}, S_{16})$ 
                                    $QR(S_1, S_6, S_{11}, S_{16})$    $QR(S_2, S_7, S_{12}, S_{13})$ 
                                    $QR(S_3, S_8, S_9, S_{14})$     $QR(S_4, S_5, S_{10}, S_{15})$ 

6   Output  $S \oplus S'$  (64 keystream bytes).
7   Increment the counter.

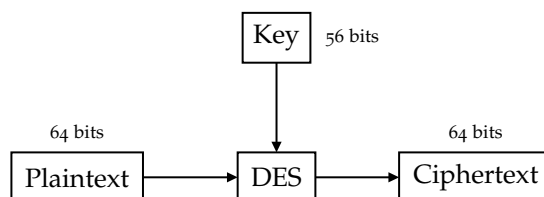
```

2.6 Block Ciphers

A block cipher is a SKES that breaks up the plaintext into blocks of a fixed length (e.g. 128 bits), and encrypts the blocks one at a time.

In contrast, a stream cipher encrypts the plaintext one character (usually a bit) at a time.

Historically important example of a block cipher: The Data Encryption Standard (DES)



Key length: 56 bits; Size of key space 2^{56} ; Block length: 64 bits.

2.6.1 Brief History of Block Ciphers

Late 1960's: Feistel network and LUCIFER designed at IBM.

1972: NBS (now NIST: National Institute of Standards and Technology) solicits proposals for encryption algorithms for the protection of computer data.

1973-1974: IBM develops DES.

1975: NSA (National Security Agency) "fixes" DES. Reduces the key length from 64 bits to 56 bits. (NSA tried for 48 bits; compromised at 56 bits.)

The National Security Agency: Founded in 1952. Budget: Classified (estimated: \$10.8 billion/year). Number of employees: Classified (30,000–40,000). Signals Intelligence (SIGINT): produce foreign intelligence information. Information Assurance (IA): protects all classified and sensitive information that is stored or sent through US government equipment. Very influential in setting US government export policy for cryptographic products (especially encryption). Canadian counterpart: Communications Security Establishment (CSE)

NSA Shenanigans

Edward Snowden. CIA/NSA contractor who disclosed in 2013 up to 200,000 NSA classified documents to the press. Currently in Russia under temporary asylum. Documents revealed the enormous capabilities of NSA and its partners (including CSE and GCHQ): “Groundbreaking cryptanalytic capabilities” Collects vast amounts of internet communications and automatically analyzes the data Directly attacks routers, switches, firewalls, etc. Installs malware on individual computers Breaks into individual computers Bruce Schneier: “The NSA is subverting the Internet and turning it into a massive surveillance tool.”

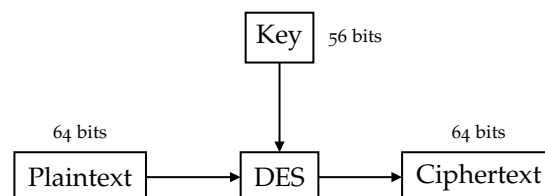
1977: DES adopted as US Federal Information Processing Standard (FIPS 46). 1981: DES adopted as a US banking standard (ANSI X3.92). 1997: NIST begins the AES (Advanced Encryption Standard) competition. 1999: 5 finalists for AES announced. 2001: Rijndael adopted for AES (FIPS 197). AES has three key lengths: 128, 192 and 256 bits. 2021: No significant weaknesses found with AES (as yet). AES uptake is rapidly increasing. However, (Triple-)DES is still deployed.

2.6.2 Some Desirable Properties of Block Ciphers

Design principles described by Claude Shannon in 1949:

- Security:
 - *Diffusion*: each ciphertext bit should depend on all plaintext bits.
 - *Confusion*: the relationship between key and ciphertext bits should be complicated.
 - *Key length*: should be small, but large enough to preclude exhaustive key search.
- Efficiency:
 - Simplicity (easier to implement and analyze).
 - High encryption and decryption rate.
 - Suitability for hardware or software.

2.6.3 The Data Encryption Standard (DES)



Key length: 56 bits; Size of key space 2^{56} ; Block length: 64 bits.

The design principles of DES are still classified, making analysis difficult. Hundreds of research papers written on the security of DES. (Triple-)DES is still deployed in practice, although its use is rapidly decreasing.

DES Problem 1: Small Key Size

Exhaustive search on key space takes 2^{56} steps and can be easily parallelized. DES challenges from RSA Security (3 known PT/CT pairs):

T	h	e		u	n	k	n	o	w	n		m	e	s	s	a	g	e		i	s	:	?	?	?	?	?	?	?
---	---	---	--	---	---	---	---	---	---	---	--	---	---	---	---	---	---	---	--	---	---	---	---	---	---	---	---	---	---

June 1997: Broken in 3 months by Internet search. 1999: Broken in 56 hours by DeepCrack machine

(1800 chips; \$250,000). 2006 Broken in 153 hours by COPACOBANA machine (\$10,000). 2012 Broken in 11.5 hours by crack.sh (\$200). RC5 64-bit challenge: July 2002: Broken in 1757 days. Participation by 331,252 individuals.

DES Problem 2: Small Block Size

If plaintext blocks are distributed “uniformly at random”, then the expected number of ciphertext blocks observed before a collision occurs is $\approx 2^{32}$ (by the birthday paradox). Hence the ciphertext reveals some information about the plaintext.

Small block length is also damaging to some authentication applications (more on this later).

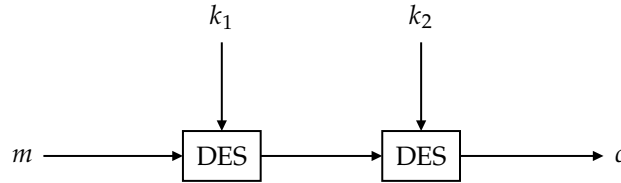
The only (substantial) weaknesses known in DES are the obvious ones: small key length and small block length.

Question: How can one construct a more secure block cipher from DES? (i.e. without changing the internals of DES.) Multiple encryption: Re-encrypt the ciphertext one or more times using independent keys. Hope that this increases the effective key length. Multiple encryption does not always result in increased security. Example: If E_π denotes the encryption function for the simple substitution cipher with key π , then is $E_{\pi_2} \circ E_{\pi_1}$ any more secure than E_π ? No, since $E_{\pi_2} \circ E_{\pi_1} = E_{\pi_2 \circ \pi_1}$.

2.6.4 Double-DES

Key is $k = (k_1, k_2)$, $k_1, k_2 \in_R \{0, 1\}^{56}$. Here $k \in_R K$ means that k is chosen uniformly and independently at random from K .

Encryption: $c = E_{k_2}(E_{k_1}(m))$. (E = DES encryption, E^{-1} = DES decryption)



Decryption: $m = E_{k_1}^{-1}(E_{k_2}^{-1}(c))$.

The Double-DES key length is $\ell = 112$, so exhaustive key search takes 2^{112} steps (infeasible). Note: Block length is unchanged.

Main idea: $c = E_{k_2}(E_{k_1}(m))$ if and only if $E_{k_2}^{-1}(c) = E_{k_1}(m)$.

Algorithm 4: Meet-In-The-Middle Attack on Double-DES

Input: 3 known PT/CT pairs $(m_1, c_1), (m_2, c_2), (m_3, c_3)$

Output: The secret key (k_1, k_2)

```

1 foreach  $h_2 \in \{0, 1\}^{56}$  do
2   | Compute  $E_{h_2}^{-1}(c_1)$ , and store  $[E_{h_2}^{-1}(c_1), h_2]$  in a table sorted by first component.
3 foreach  $h_1 \in \{0, 1\}^{56}$  do
4   | Compute  $E_{h_1}(m_1)$ .
5   | Search for  $E_{h_1}(m_1)$  in the table. (we say that  $E_{h_1}(m_1)$  matches table entry  $[E_{h_2}^{-1}(c_1), h_2]$  if
6     |  $E_{h_2}^{-1}(c_1) = E_{h_1}(m_1)$ .)
7   | foreach match  $[E_{h_2}^{-1}(c_1), h_2]$  in the table do
8     |   | if  $E_{h_2}(E_{h_1}(m_2)) = c_2$  then
9     |   |   | if  $E_{h_2}(E_{h_1}(m_3)) = c_3$  then
9     |   |   |   | Output  $(h_1, h_2)$  and STOP.
  
```

Number of known plaintext/ciphertext pairs needed for unique key determination:

Question: Let E be a block cipher with key space $K = \{0,1\}^\ell$, and plaintext and ciphertext space $\{0,1\}^L$.

Let $k' \in K$ be the secret key chosen by Alice and Bob, and let $(m_i, c_i), 1 \leq i \leq t$, be known plaintext/-ciphertext pairs, where the plaintext m_i are distinct. (Note that $c_i = E_{k'}(m_i)$ for all $1 \leq i \leq t$)

Then how large should t be to ensure (with probability very close to 1) that there is only one key $k \in K$ such that $E_k(m_i) = c_i$ for all $1 \leq i \leq t$?

Answer: Select t so that $FK \approx 0$.

For each $k \in K$, then encryption function $E_k : \{0,1\}^L \rightarrow \{0,1\}^L$ is a permutation.

We make the heuristic assumption that for each $k \in K$, E_k is a random function (i.e., a randomly selected function). This assumption is certainly false since E_k is not random, and because a random function is almost certainly not a permutation. Nonetheless, it turns out that the assumption is good enough for our analysis.

Now, fix $k \in K, k \neq k'$. The probability that $E_k(m_i) = c_i$ for all $1 \leq i \leq t$ is

$$\underbrace{\frac{1}{2^L} \cdot \frac{1}{2^L} \cdots \frac{1}{2^L}}_t = \frac{1}{2^{Lt}}.$$

Thus the expected number of false keys $k \in K$ (not including k') for which $E_k(m_i) = c_i$ for all $1 \leq i \leq t$ is

$$FK = \frac{2^\ell - 1}{2^{Lt}}.$$

Meet-In-The-Middle Attack on Double DES

Let E be the DES encryption function, so Double-DES encryption is $c = E_{k_2}(E_{k_1}(m))$.

1. If $\ell = 112, L = 64, t = 3$, then $FK \approx 1/2^{80} \approx 0$. Thus if a Double-DES key (h_1, h_2) is found for which $E_{h_2}(E_{h_1}(m_i)) = c_i$ for $i = 1, 2, 3$, then with very high probability we have $(h_1, h_2) = (k_1, k_2)$.
2. If $\ell = 112, L = 64, t = 1$, then $FK \approx 2^{48}$. Thus the expected number of Double-DES keys (h_1, h_2) for which $E_{h_2}(E_{h_1}(m_1)) = c_1$ is $\approx 2^{48}$.
3. If $\ell = 112, L = 64, t = 2$, then $FK \approx 1/2^{16}$. Thus the expected number of Double-DES keys (h_1, h_2) for which $E_{h_2}(E_{h_1}(m_1)) = c_1$ and $E_{h_2}(E_{h_1}(m_2)) = c_2$ is $\approx 1/2^{16}$.

Analysis:

- Number of DES operations is $\approx 2^{56} + 2^{56} + 2 \cdot 2^{48} \approx 2^{57}$. (We are not counting the time to do the sorting and searching)
- Space requirements: $2^{56}(64 + 56)$ bits $\approx 1,080,863$ Tbytes.

Time-memory tradeoff. [Exercise] The attack can be modified to decrease the storage requirements at the expense of time: Time: 2^{56+s} steps; memory: 2^{56-s} units, $1 \leq s \leq 55$.

Conclusions: Double-DES has the same effective key length as DES. Double-DES is not much more secure than DES.

Storage Units

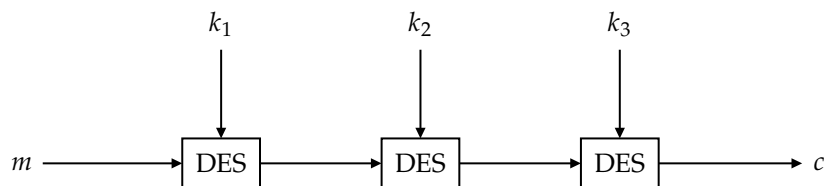
- 10^3 bytes = 1 Kbyte (kilo) $\approx 2^{10}$ bytes
- 10^3 Kbytes = 1 Mbyte (mega) $\approx 2^{20}$ bytes

- 10^3 Mbytes = 1 Gbyte (giga) $\approx 2^{30}$ bytes
- 10^3 Gbytes = 1 Tbyte (tera) $\approx 2^{40}$ bytes
- 10^3 Tbytes = 1 Pbyte (peta) $\approx 2^{50}$ bytes
- 10^3 Pbytes = 1 Ebyte (exa) $\approx 2^{60}$ bytes

2.6.5 Triple-DES

Triple-DES. Key is $k = (k_1, k_2, k_3)$, $k_1, k_2, k_3 \in_R \{0, 1\}^{56}$.

Encryption: $c = E_{k_3}(E_{k_2}(E_{k_1}(m)))$ where E = DES encryption, E^{-1} = DES decryption



Decryption: $m = E_{k_1}^{-1}(E_{k_2}^{-1}(E_{k_3}^{-1}(c)))$.

Key length of Triple-DES is $\ell = 168$, so exhaustive key search takes 2^{168} steps (infeasible)

Meet-in-the-middle attack takes $\approx 2^{112}$ steps. [Exercise]

So, the effective key length of Triple-DES against exhaustive key search is ≈ 112 bits.

No proof that Triple-DES is more secure than DES.

Note: Block length is unchanged.

Triple-DES is still deployed: especially by some financial institutions, where its use can be expected to continue until 2030.

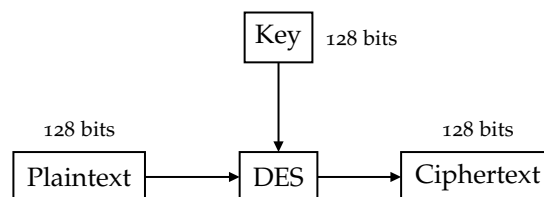
2.6.6 The Advanced Encryption Standard (AES)

<http://www.nist.gov/aes>

1997: Call issued for AES candidate algorithms.

Requirements:

- Key lengths: 128, 192 and 256 bits.
- Block length: 128 bits.
- Efficient on both hardware and software platforms.
- Availability on a worldwide, non-exclusive, royalty-free basis.



The AES Process: 1998: 15 submissions in Round 1. 1999: 5 finalists selected by NIST: MARS, RC6, Rijndael, Serpent, Twofish. 1999: NSA performed a hardware efficiency comparison. 2000: Rijndael was selected. 2001: The AES standard is officially adopted (FIPS 197). Rijndael is an example of a

substitution-permutation network. 2021: No attacks have been found on AES that are (significantly) faster than exhaustive key search.

2.6.7 Substitution-Permutation Networks

A substitution-permutation network (SPN) is an iterated block cipher where a round consists of a substitution operation followed by a permutation operation.

- n : the block length.
- ℓ : the key length.
- h : the number of rounds.
- A fixed invertible function $S : \{0,1\}^b \rightarrow \{0,1\}^b$, where b is a divisor of n .
- A fixed permutation P on $\{1,2,\dots,n\}$.
- A key scheduling algorithm that determines subkeys $k_1, k_2, \dots, k_h, k_{h+1}$ from a key k .

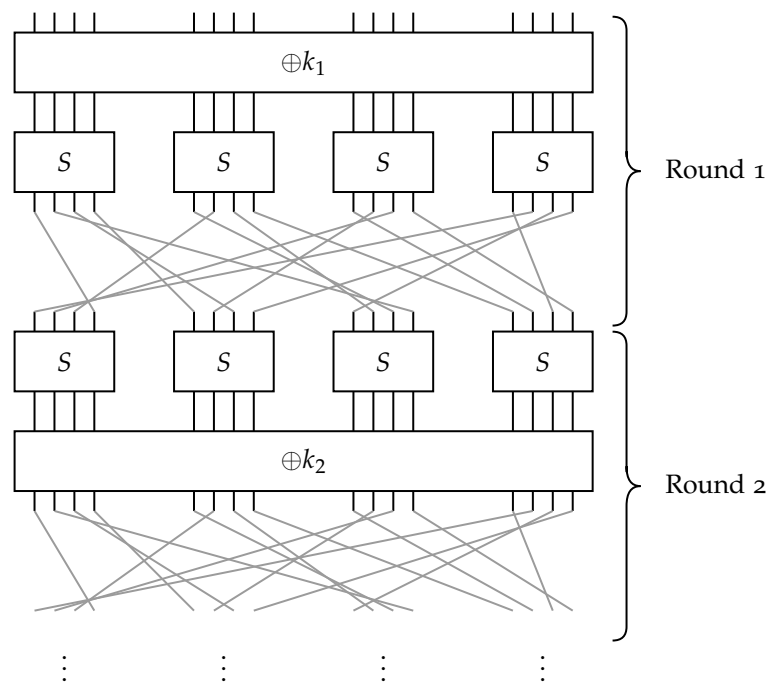
Note: n, ℓ, h, S, P and the key scheduling algorithm are public. The only secret in AES is the key k that is selected.

Here is a description of encryption:

Algorithm 5: Encryption of Substitution-Permutation Networks

```

1  $A \leftarrow \text{plaintext}$ 
2 for  $i = 1..h$  do
3    $A \leftarrow A \oplus k_i$  // XOR
4    $A \leftarrow S(A)$  // Substitution
5    $A \leftarrow P(A)$  // Permutation
6  $A \leftarrow A \oplus k_{h+1}$ 
7 ciphertext  $\leftarrow A$ 
```



2.6.8 AES

AES is an SPN, where the permutation operation is replaced by two invertible linear transformations. All operations are byte oriented (e.g., $b = 8$ so the S-box maps 8-bits to 8-bits). This allows AES to be efficiently implemented on software platforms. The block length of AES is $n = 128$ bits. Each subkey is 128 bits. AES accepts three different key lengths. The number of rounds h depends on the key length:

cipher	key length ℓ	h
AES-128	128	10
AES-192	192	12
AES-256	256	14

AES Round Operations

Each round updates a variable called State which consists of a 4×4 array of bytes (note: $4 \cdot 4 \cdot 8 = 128$, the block length). State is initialized with the plaintext:

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

\leftarrow plaintext

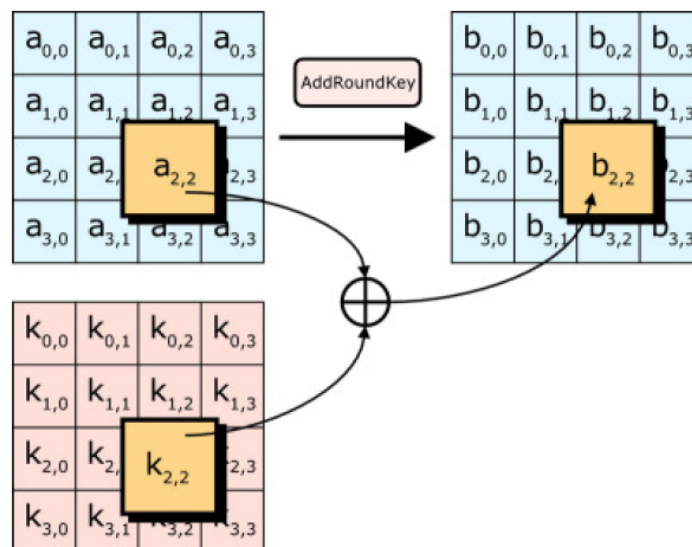
After h rounds are completed, a final subkey is XOR-ed with State, the result being the ciphertext.

The AES round function uses four invertible operations:

1. AddRoundKey (key mixing)
2. SubBytes (S-box)
3. ShiftRows (permutation)
4. MixColumns (linear transformation).

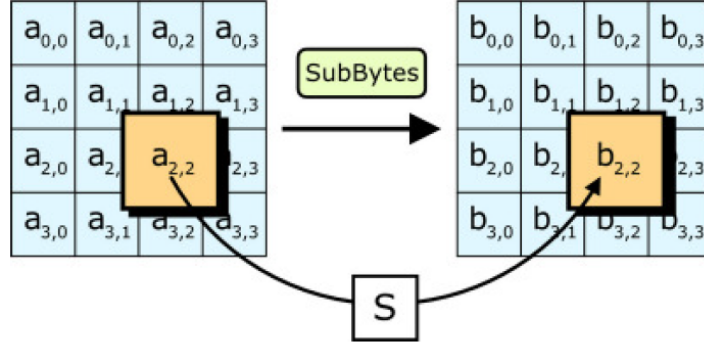
Add Round Key

Bitwise-XOR each byte of State with the corresponding byte of the subkey.



Substitute Bytes

Take each byte in State and replace it with the output of the S-box.



$S : \{0,1\}^8 \rightarrow \{0,1\}^8$ is a fixed, public, invertible function. The S-box is a non-linear function.

The Finite Field $GF(2^8)$

The elements of the finite field $GF(2^8)$ are the polynomials of degree at most 7 in $\mathbb{Z}_2[y]$, with addition and multiplication performed modulo the irreducible polynomial $f(y) = y^8 + y^4 + y^3 + y + 1$.

Interpret an 8-bit string $a = a_7a_6a_5a_4a_3a_2a_1a_0$ as coefficients of the polynomial $a(y) = a_7y^7 + a_6y^6 + a_5y^5 + \dots + a_1y + a_0$ and vice versa.

Example:

Let $a = 11101100 = \text{ec}$ and $b = 00111011 = 3\text{b}$, so $a(y) = y^7 + y^6 + y^5 + y^3 + y^2$ and $b(y) = y^5 + y^4 + y^3 + y + 1$.

1. Addition: $a(y) + b(y) = y^7 + y^6 + y^5 + y^3 + y^2 + y + 1$, so $\text{ec} + 3\text{b} = \text{d7}$.
2. Multiplication: $a(y) \cdot b(y) = y^{12} + y^{10} + y^8 + y^4 + y^2$, which leaves a remainder of $r(y) = y^7 + y^6 + y^3$ upon division by $f(y)$. Hence $a \cdot b = 11001000$ in $GF(2^8)$, or $\text{ec} \cdot 3\text{b} = \text{c8}$.
3. Inversion: $\text{ec}^{-1} = 5\text{f}$ since $\text{ec} \cdot 5\text{d} = 01$.

S-box Definition

Let $p \in \{0,1\}^8$, and consider p as an element of $GF(2^8)$.

1. Let $q = p^{-1}$ if $p \neq 0$, and $q = p$ if $p = 0$.
2. Define $q = (q_7q_6q_5q_4q_3q_2q_1q_0)$.
3. Compute

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Then $S(p) = r = (r_7r_6r_5r_4r_3r_2r_1r_0)$.

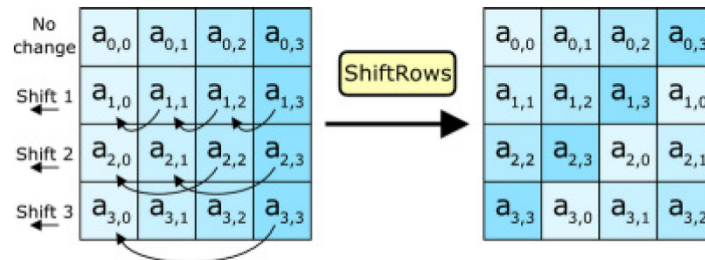
The AES S-box

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	fo	ad	d4	a2	af	9c	a4	72	co
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	ao	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	do	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	oc	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	ob	db
a	eo	32	3a	oa	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	o8
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	oe	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	od	bf	e6	42	68	41	99	2d	of	bo	54	bb	16

For example, $S(a8) = c2$.

Shift Rows

Permute the bytes of State by applying a cyclic shift to each row.



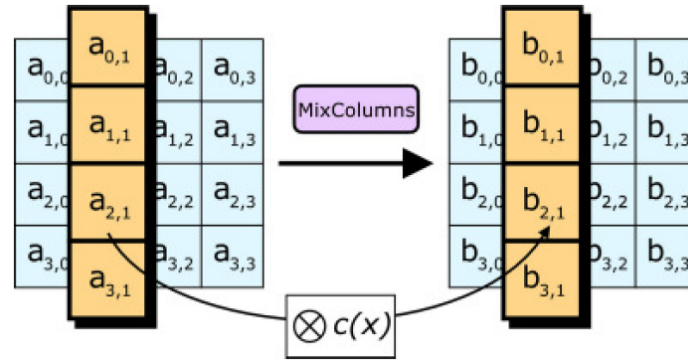
Mix Columns

1. Read column i of State as a polynomial:

$$(a_{0,i}, a_{1,i}, a_{2,i}, a_{3,i}) = a_{0,i} + a_{1,i}x + a_{2,i}x^2 + a_{3,i}x^3$$

(interpret the coefficients as elements of the finite field $GF(2^8)$).

2. Multiply this polynomial with the constant polynomial $c(x) = 03 \cdot x^3 + 01 \cdot x^2 + 01 \cdot x + 02$ and reduce modulo $x^4 - 1$. This gives a new polynomial: $b_{0,i} + b_{1,i}x + b_{2,i}x^2 + b_{3,i}x^3$



The $\otimes c(x)$ Operation

- Let $a(x) = a_0 + a_1x + a_2x^2 + a_3x^3$, where each $a_i \in GF(2^8)$.
- Let $c(x) = 02 + 01x + 01x^2 + 03x^3$, where 01, 02, 03 are elements in $GF(2^8)$ (written in hexadecimal).
- To compute $a(x) \otimes c(x)$:
 - Compute $d(x) = a(x) \times c(x)$ (polynomial multiplication, where coefficient arithmetic is in $GF(2^8)$).
 - Divide by $x^4 - 1$ to find the remainder polynomial $r(x)$ (equivalently, replace x^4 by 1, x^5 by x , and x^6 by x^2).
 - Then $a(x) \otimes c(x) = r(x)$.

For example, let $a(x) = d0f112bb = d0 + f1x + 12x^2 + bbx^3$. Then $a(x) \otimes c(x) = 1a + a4x + d3x^2 + e5x^3 = 1aa4d3e5$.

AES Encryption

From the key k derive $h + 1$ subkeys k_0, k_1, \dots, k_h .

Here is a description of encryption:

Algorithm 6: AES Encryption

```

1 State  $\leftarrow$  plaintext
2 State  $\leftarrow$  State  $\oplus k_0$ 
3 for  $i = 1..h - 1$  do
4   | State  $\leftarrow$  SubBytes(State)
5   | State  $\leftarrow$  ShiftRows(State)
6   | State  $\leftarrow$  MixColumns(State)
7   | State  $\leftarrow$  State  $\oplus k_i$ 
8 State  $\leftarrow$  SubBytes(State)
9 State  $\leftarrow$  ShiftRows(State)
10 State  $\leftarrow$  State  $\oplus k_h$ 
11 ciphertext  $\leftarrow$  State

```

Note that in the final round, **MixColumns** is not applied. This helps make decryption more similar to encryption, and thus is useful when implementing AES. [Details omitted]

AES Decryption

From the key k derive $h + 1$ subkeys k_0, k_1, \dots, k_h .

Here is a description of decryption:

Algorithm 7: AES Decryption

```

1 State  $\leftarrow$  ciphertext
2 State  $\leftarrow$  State  $\oplus k_h$ 
3 State  $\leftarrow$  InvShiftRows(State)
4 State  $\leftarrow$  InvSubBytes(State)
5 for  $i = h - 1 \dots 1$  do
6   State  $\leftarrow$  State  $\oplus k_i$ 
7   State  $\leftarrow$  InvMixColumns(State)
8   State  $\leftarrow$  InvShiftRows(State)
9   State  $\leftarrow$  InvSubBytes(State)
10 State  $\leftarrow$  State  $\oplus k_0$ 
11 plaintext  $\leftarrow$  State

```

InvMixColumns is multiplication by $d(x) = 0e + 09x + 0dx^2 + 0bx^3$ modulo $x^4 - 1$.

AES Key Schedule (for 128-bit keys)

- For 128-bit keys, AES has 10 rounds, so we need 11 subkeys.
- The first subkey $k_0 = (r_0, r_1, r_2, r_3)$ is the actual AES key.
- The second subkey is $k_1 = (r_4, r_5, r_6, r_7)$.
- The third subkey is $k_2 = (r_8, r_9, r_{10}, r_{11})$.
- ...
- The eleventh subkey is $k_{10} = (r_{40}, r_{41}, r_{42}, r_{43})$.

AES Key Schedule Functions f_i

The functions $f_i : \{0, 1\}^{32} \rightarrow \{0, 1\}^{32}$ are defined as follows:

1. The input is divided into 4 bytes: (a, b, c, d) .
2. Left-rotate the bytes: (b, c, d, a) .
3. Apply the AES S-box to each byte: $(S(b), S(c), S(d), S(a))$.
4. XOR the leftmost byte with the constant ℓ_i , and output the result: $(S(b) \oplus \ell_i, S(c), S(d), S(a))$.

The constants ℓ_i :

i	ℓ_i	i	ℓ_i	i	ℓ_i	i	ℓ_i	i	ℓ_i
1	0x01	2	0x02	3	0x04	4	0x08	5	0x10
6	0x20	7	0x40	8	0x80	9	0x1b	10	0x36

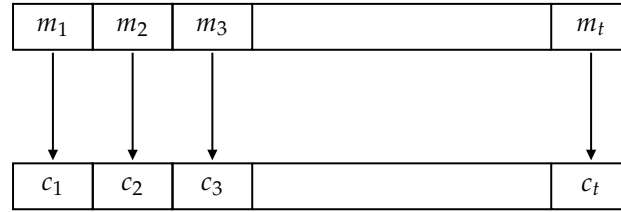
2.6.9 Block Cipher Modes of Operation

In practice, one might need to encrypt a large quantity of data. Plaintext message is $m = m_1, m_2, \dots, m_t$, where each m_i is an L -bit block.

Question: How should we use a block cipher $E_k : \{0,1\}^L \rightarrow \{0,1\}^L$ to encrypt m ?

Electronic Codebook (ECB) Mode

Encrypt blocks independently, one at a time: $c = c_1, c_2, \dots, c_t$, where $c_i = E_k(m_i)$.

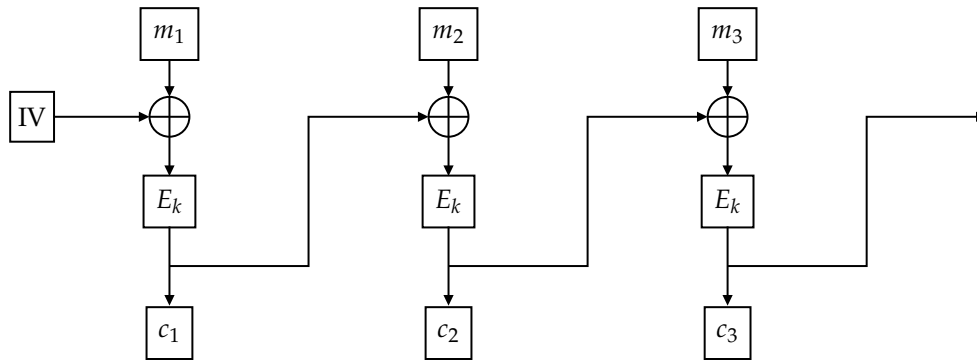


Decryption: $m_i = E_k^{-1}(c_i)$, $i = 1, 2, \dots, t$.

Drawback: Identical plaintexts result in identical ciphertexts (under the same key), and thus ECB encryption is not (semantically) secure against chosen-plaintext attacks. [Why?]

Cipher Block Chaining (CBC) Mode

Encryption: Select $c_0 \in_R \{0,1\}^L$ (c_0 is a random non-secret IV). Then compute $c_i = E_k(m_i \oplus c_{i-1})$, $i = 1, 2, \dots, t$.



Ciphertext is $(c_0, c_1, c_2, \dots, c_t)$.

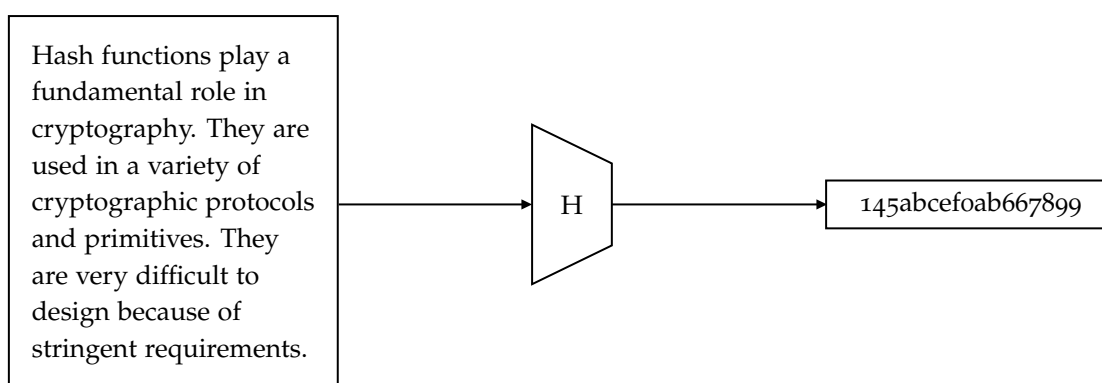
Decryption: $m_i = E_k^{-1}(c_i) \oplus c_{i-1}$, $i = 1, 2, \dots, t$.

Identical plaintexts with different IVs result in different ciphertexts and for this reason CBC encryption is (semantically) secure against chosen-plaintext attacks (for a well chosen block cipher E).

Hash Functions

Hash functions play a fundamental role in cryptography. They are used in a variety of cryptographic primitives and protocols. They are very difficult to design because of very stringent security and performance requirements. The main candidates available today are: SHA-1; SHA-2 family: SHA-224, SHA-256, SHA-384, SHA-512

What is a Hash Function?



See

- <http://www.xorbin.com/tools/md5-hash-calculator>
- <http://www.xorbin.com/tools/sha1-hash-calculator>
- <http://www.xorbin.com/tools/sha256-hash-calculator>

hash function

A **hash function** is a mapping H such that:

1. H maps binary messages of arbitrary lengths $\leq L$ to outputs of a fixed length n : $H : \{0,1\}^{\leq L} \rightarrow \{0,1\}^n$. (L is large, e.g., $L = 2^{64}$; n is small, e.g., $n = 256$)
2. $H(x)$ can be efficiently computed for all $x \in \{0,1\}^{\leq L}$.

H is called an n -bit hash function. $H(x)$ is called the hash or message digest of x .

Note:

The description of a hash function is public. There are no secret keys.

For simplicity, we will usually write $\{0,1\}^*$ instead of $\{0,1\}^{\leq L}$.

Toy Hash Function

$$H : \{0,1\}^{\leq 4} \rightarrow \{0,1\}^2$$

x	$H(x)$	x	$H(x)$	x	$H(x)$	x	$H(x)$
0	00	1	01				
00	11	01	01	10	01	11	00
000	00	001	10	010	11	011	11
100	11	101	01	110	01	111	10
0000	00	0001	11	0010	11	0011	00
0100	01	0101	10	0110	10	0111	01
1000	11	1001	01	1010	00	1011	01
1100	10	1101	00	1110	00	1111	11

(00, 1000) is a collision.

1001 is a preimage of 01.

10 is a second preimage of 1011.

Example: SHA-256

$$\text{SHA-256: } \{0,1\}^* \rightarrow \{0,1\}^{256}$$

SHA-256("Hello there") =

0x4e47826698bb4630fb4451010062fadbf85d61427cbdfaed7ad0f23f239bed89

SHA-256("Hello There") =

0xabf5dacd019d2229174f1daa9e62852554ab1b955fe6ae6bbbb214bab611f6f5

SHA-256("Welcome to CO 487") =

0x819ba4b1e568e516738b48d15b568952d4a35ea73f801c873907d3ae1f5546fb

SHA-256("Welcome to CO 687") =

0x404fb0ee527b8f9f01c337e915e8beb6e03983cfd9544296b8cf0e09c9d8753d

Davies-Meyer hash function. Let E_k be an m -bit block cipher with n -bit key k . Let IV be a fixed m -bit initializing value.

To compute $H(x)$, do:

- Break up $x \parallel 1$ into n -bit blocks: $\bar{x} = x_1, x_2, \dots, x_t$, padding out the last block with 0 bits if necessary.
- Define $H_0 = IV$.
- Compute $H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}$ for $i = 1, \dots, t$.
- Define $H(x) = H_t$.

Hash functions are the Swiss Army knife of cryptography. Hash functions are used for all kinds of applications they were not designed for. One reason for this widespread use of hash functions is speed.

Preimage Resistance (PR)

$$H : \{0,1\}^* \rightarrow \{0,1\}^n.$$

Preimage resistance hash function

Given a hash value $y \in_R \{0,1\}^n$, it is computationally infeasible to find (with non-negligible probability of success) any $x \in \{0,1\}^*$ such that $H(x) = y$. x is called a preimage of y .

Note: $x \in_R S$ means that x is chosen independently and uniformly at random from S .

Password protection on a multi-user computer system:

- Server stores $(\text{userid}, H(\text{password}))$ in a password file. Thus, if an attacker gets a copy of the password file, she does not learn any passwords.
- Requires preimage-resistance.

2nd Preimage Resistance (2PR)

$$H : \{0,1\}^* \rightarrow \{0,1\}^n.$$

Second-preimage resistance hash function

Given an input $x \in \{0,1\}^*$, it is computationally infeasible to find (with non-negligible probability of success) a second input $x' \in \{0,1\}^*$, $x' \neq x$, such that $H(x') = H(x)$.

Modification Detection Codes (MDCs).

- To ensure that a message m is not modified by unauthorized means, one computes $H(m)$ and protects $H(m)$ from unauthorized modification.
- e.g. virus protection. Requires 2nd preimage resistance.

Other Applications of Hash Functions

1. Message Authentication Codes (HMAC). [More on this later]
2. Pseudorandom bit generation: Distilling random bits s from several “pseudorandom” sources x_1, \dots, x_t . Output $s = H(x_1, \dots, x_t)$.
3. Key derivation function (KDF): Deriving a cryptographic key from a shared secret. [More on this later]
4. Proof-of-work in cryptocurrencies (Bitcoin): [More on this later]
5. Quantum-safe signature schemes: [More on this later]

Typical Cryptographic Requirements

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

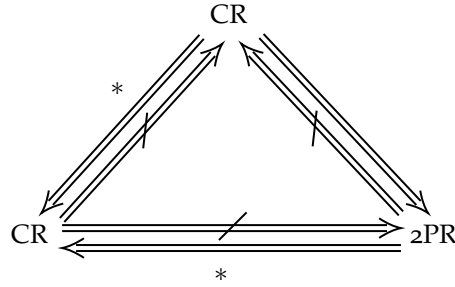
Preimage resistance, 2nd preimage resistance.

Collision resistance

It is computationally infeasible to find (with non-negligible probability of success) two distinct inputs $x, x' \in \{0, 1\}^*$ such that $H(x) = H(x')$.

Relationships between PR, 2PR, CR

Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ be a hash function.



* for somewhat uniform hash functions

1. CR implies 2PR

Proof:

Suppose that H is not 2PR.

Select $x \in_R \{0, 1\}^*$. Since H is not 2PR, we can efficiently find $x' \in \{0, 1\}^*$, $x' \neq x$, with $H(x') = H(x)$. Thus we have efficiently found a collision (x, x') for H . Hence H is not CR. \square

Note that this proof establishes the contrapositive statement.

2. 2PR does not guarantee CR

Proof:

Suppose that $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ is 2PR.

Consider $\bar{H} : \{0, 1\}^* \rightarrow \{0, 1\}^n$ defined by:

$$\bar{H}(x) = \begin{cases} H(0), & \text{if } x = 1, \\ H(x), & \text{if } x \neq 1. \end{cases} \quad \bar{H}(1) = H(0) = \bar{H}(0)$$

Suppose that \bar{H} is not 2PR. So, given $x \in_R \{0, 1\}^*$, we can efficiently find $x' \in \{0, 1\}^*$, $x' \neq x$, with $\bar{H}(x') = \bar{H}(x)$. Now, if $x' \neq 1$, then $\bar{H}(x') = \bar{H}(x') = H(x)$; while if $x' = 1$, then $\bar{H}(x') = \bar{H}(1) = H(0) = H(x)$. In either case, we have found a second preimage for x with respect to H . So, \bar{H} must be 2PR.

Also, \bar{H} is not CR, since $(0, 1)$ is a collision for \bar{H} . \square

3. CR does not guarantee PR

Proof:

Suppose that $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$ is CR.

Consider $\bar{H} : \{0, 1\}^* \rightarrow \{0, 1\}^{n+1}$ defined by:

$$\bar{H}(x) = \begin{cases} 1 \parallel x, & \text{if } x \in \{0, 1\}^n, \\ 0 \parallel H(x), & \text{if } x \notin \{0, 1\}^n. \end{cases}$$

Then \bar{H} is CR (since H is).

And, \bar{H} is not PR since preimages can be efficiently found for at least half of all $y \in \{0,1\}^{n+1}$. \square

However, if H is “somewhat uniform” (i.e., all hash values have roughly the same number of preimages), then CR does not imply PR.

Proof:

Suppose H is not PR. Select $x \in_R \{0,1\}^*$ and compute $y = H(x)$. Since H is not PR, we can efficiently find a preimage x' of y . Since H is somewhat uniform, we expect that y has many preimages, and thus $x' \neq x$ with very high probability. Thus, (x, x') is a collision for H that we have efficiently found, so H is not CR. \square

Note that we shall henceforth assume that hash functions are somewhat uniform.

4. PR does not guarantee 2PR

Proof:

Suppose $H : \{0,1\}^* \rightarrow \{0,1\}^n$ is PR.

Define $\bar{H} : \{0,1\}^* \rightarrow \{0,1\}^n$ by

$$\bar{H}(x_1, x_2, \dots, x_t) = H(0, x_2, \dots, x_t).$$

Then \bar{H} is PR, but \bar{H} is not 2PR. \square

5. 2PR implies PR (for somewhat uniform H)

Proof:

Exercise. \square

3.1 Generic Attacks

generic attack

A **generic attack** on a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$ does not exploit any properties the specific hash function may have.

In the analysis of a generic attack, we view H as a random function in the sense that for each $x \in \{0,1\}^*$, we assume that the value $y = H(x)$ was chosen by selecting $y \in_R \{0,1\}^n$.

From a security point of view, a random function is an ideal hash function. However, random functions are not suitable for practical applications because they cannot be compactly stored.

Generic Attack for Finding Preimages

$H : \{0,1\}^* \rightarrow \{0,1\}^n$.

Given $y \in_R \{0,1\}^n$, repeatedly select arbitrary $x \in \{0,1\}^*$ until $H(x) = y$.

Expected number of steps is $\approx 2^n$. (Here, a ‘step’ is a hash function evaluation.)

This attack is infeasible if $n \geq 128$.

Note:

It has been proven that this generic attack for finding preimages is optimal, i.e., no faster generic attack exists.

Of course, for a specific hash function, there might exist a preimage finding algorithm that is faster than the generic attack.

Generic Attack for Finding Collisions

Select arbitrary $x \in \{0, 1\}^*$ and store $(H(x), x)$ in a table sorted by first entry. Continue until a collision is found.

Expected number of steps: $\sqrt{\pi 2^n / 2} \approx \sqrt{2^n}$ (by birthday paradox). (Here, a “step” is a hash function evaluation.)

This attack is infeasible if $n \geq 256$.

It has been proven that this generic attack for finding collisions is optimal in terms of the number of hash function evaluations.

Expected space required: $\sqrt{\pi 2^n / 2} \approx \sqrt{2^n}$.

If $n = 128$:

- Expected running time: 2^{64} steps. [feasible]
- Expected space required: 7×10^8 Tbytes. [infeasible]

VW Parallel Collision Search

VW: Van Oorschot & Wiener (1993)

Expected number of steps: $\sqrt{2^n}$

Expected space required: negligible.

Easy to parallelize: m -fold speedup with m processors.

The collision-finding algorithm can easily be modified to find “meaningful” collisions. [see optional readings]

Conclusion: If collision resistance is desired, then use an n -bit hash function with $n \geq 256$

Parallel collision search (VW Method)

Problem: Find a collision for $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$.

Assumption: H is a random function.

VW: Paul van Oorschot, Michael Wiener (1993)

Notation: Let $N = 2^n$. Define a sequence $\{x_i\}_{i \geq 0}$ by $x_0 \in_R \{0, 1\}^n$, $x_i = H(x_{i-1})$ for $i \geq 1$. Let j be the smallest index for which $x_j = x_i$ for some $i < j$; such a j must exist. Then $x_{j+\ell} = x_{i+\ell}$ for all $\ell \geq 1$. By the birthday paradox, $E[j] \approx \sqrt{\frac{\pi N}{2}} \approx \sqrt{N}$. In fact, $E[i] \approx \frac{1}{2}\sqrt{N}$ and $E[j-1] \approx \frac{1}{2}\sqrt{N}$.

Now, $i \neq 0$ with overwhelming probability; in that event, (x_{i-1}, x_{j-1}) is a collision for H .

How to find (x_{i-1}, x_{j-1}) without using much storage?

Answer: Store only distinguished points.

Select an easily testable distinguishing property for elements of $\{0, 1\}^n$ (e.g., leading 32 bits are all 0). Let θ be the proportion of elements of $\{0, 1\}^n$ that are distinguished.

VW algorithm Compute x_0, x_1, x_2, \dots and only store the points in the sequence that are distinguished.

Algorithm 8: VW collision finding

```

// Stage 1 (detecting a collision)
1 Select  $x_0 \in_R \{0, 1\}^n$ 
2 Store  $(x_0, 0, -)$  in a sorted table
3  $LP \leftarrow x_0$  //  $LP = \text{last point stored}$ 
4 for  $d = 1, 2, 3, 4, \dots$  do
5   Compute  $x_d = H(x_{d-1})$ 
6   if  $x_d$  is distinguished then
7     If  $x_d$  is already in table, say  $x_d = x_b$  where  $b < d$  then go to stage 2.
8     Store  $(x_d, d, LP)$  in a table.
9      $LP \leftarrow x_d$ 
// Stage 2 (finding a collision)
10 Set  $\ell_1 \leftarrow b - d, \ell_2 \leftarrow d - c$ .
11 Suppose  $\ell_1 \geq \ell_2$ ; set  $k \leftarrow \ell_1 - \ell_2$ 
12 Compute  $x_{a+1}, x_{a+2}, \dots, x_{a+R}$ .
13 for  $m = 1, 2, 3, \dots$  do
14   Compute  $(x_{a+k+m}, x_{c+m})$ 
15 until  $x_{a+k+m} = x_{c+m}$ 
16 The collision is  $(x_{a+k+m-1}, x_{c+m-1})$ 

```

Analysis

- Stage 1: Expected # of H -evaluations is

$$\underbrace{\sqrt{\frac{\pi N}{2}}}_{\text{collision occurs}} + \underbrace{\frac{1}{\theta}}_{\text{collision is detected}} \approx \sqrt{N} + \frac{1}{\theta}$$

- Stage 2: Expected # of H -evaluations is $\leq \frac{3}{\theta}$ (see optional readings)

Overall expected running time: $\sqrt{N} + \frac{4}{\theta}$

Storage: $\approx \theta\sqrt{N}(3N)$ bits (each table entry has bitlength $3n$)

Example:

Let $n = 128$. take $\theta = \frac{1}{2^{32}}$. Then the expected time of VW collision search is 2^{64} H -evaluations (feasible), and the expected storage is 241 Gbytes (free).

Parallelizing VW collision search (m processors)

Run a copy of VW on each of the m processors. Report distinguished points to a central server.

Analysis:

- Expected time $\approx \sqrt{N}/m + 4/\theta$
- Expected storage $\approx \theta\sqrt{N}(3n)$ bits

Note:

Factor- m speedup.

No communication between processors.

■ Occasional communication with central server

3.2 Iterated Hash Functions (Merkle Meta-Method)

Components:

- Fixed initializing value $IV \in \{0, 1\}^n$,
- Compression function $f : \{0, 1\}^{n+r} \rightarrow \{0, 1\}^n$ (efficiently computable).

To compute $H(x)$ where x has bitlength $b < 2^r$ do:

- Break up x into r -bit blocks: $\bar{x} = x_1, x_2, \dots, x_t$, padding out the last block with 0 bits if necessary.
- Define x_{t+1} , the length-block, to hold the right-justified binary representation of b .
- Define $H_0 = IV$.
- Compute $H_i = f(H_{i-1}, x_i)$ for $i = 1, 2, \dots, t + 1$.
- Define $H(x) = H_{t+1}$.

The H_i 's are called chaining variables.

3.2.1 Collision Resistance of Iterated Hash Functions

Theorem 3.1: Merkle

If the compression function f is collision resistant, then the hash function H is also collision resistant.

Merkle's theorem reduces the problem of designing collision-resistant hash functions to that of designing collision-resistant compression functions.

3.3 Provable Security

A major theme of cryptographic research is to formulate precise security definitions and assumptions, and then prove that a cryptographic protocol is secure. A proof of security is certainly desirable since it rules out the possibility of attacks being discovered in the future. However, it isn't always easy to assess the practical security assurances (if any) that a security proof provides.

- The assumptions might be unrealistic, or false, or circular.
- The security proof might be fallacious.
- The security model might not account for certain kinds of realistic attacks.
- The security proof might be asymptotic.
- The security proof might have a large tightness gap.

Optional reading: <http://anotherlook.ca>

Proof:

Suppose that H is not collision resistant. Then we can efficiently find $x, x' \in \{0, 1\}^*$, $x \neq x'$, with $H(x) = H(x')$.

Let

$$\begin{aligned}\bar{x} &= x_1, x_2, \dots, x_t, & b &= \text{bitlength}(x), & x_{t+1} &= \text{length block} \\ \bar{x}' &= x'_1, x'_2, \dots, x'_t, & b' &= \text{bitlength}(x'), & x'_{t'+1} &= \text{length block}.\end{aligned}$$

We efficiently compute:

$$\begin{aligned}H_0 &= IV \\ H_1 &= f(H_0, x_1) \\ H_2 &= f(H_1, x_2) \\ &\vdots \\ H_{t-1} &= f(H_{t-2}, x_{t-1}) \\ H_t &= f(H_{t-1}, x_t) \\ H_{t+1} &= f(H_t, x_{t+1})\end{aligned}$$

and

$$\begin{aligned}H'_0 &= IV \\ H'_1 &= f(H'_0, x'_1) \\ H'_2 &= f(H'_1, x'_2) \\ &\vdots \\ H'_{t'-1} &= f(H'_{t'-2}, x'_{t'-1}) \\ H'_{t'} &= f(H'_{t'-1}, x'_{t'}) \\ H'_{t'+1} &= f(H'_{t'}, x'_{t'+1})\end{aligned}$$

Since $H(x) = H(x')$, we have $H_{t+1} = H'_{t'+1}$.

Now if $b \neq b'$, then $x_{t+1} \neq x'_{t'+1}$. Thus $(H_t, x_{t+1}), (H'_{t'}, x'_{t'+1})$ is a collision for f that we have efficiently found. Hence f is not collision resistant.

Suppose $b = b'$. Then $t = t'$ and $x_{t+1} = x'_{t'+1}$. Let i be the largest index, $0 \leq i \leq t$, for which

$$(H_i, x_{i+1}) \neq (H'_i, x'_{i+1}).$$

Such an i must exist since $x \neq x'$. Then

$$H_{i+1} = f(H_i, x_{i+1}) = f(H'_i, x'_{i+1}) = H'_{i+1},$$

so $(H_i, x_{i+1}), (H'_i, x'_{i+1})$ is a collision for f that we have efficiently found. Hence f is not collision resistant. \square

3.4 MDx-Family of Hash Functions

MDx is a family of iterated hash functions. MD4 was proposed by Ron Rivest in 1990. MD4 has 128-bit outputs. Wang et al. (2004) found collisions for MD4 by hand. Leurent (2008) discovered an algorithm for finding MD4 preimages in 2^{102} steps.

3.4.1 MD5 Hash Function

MD5 is a strengthened version of MD4. Designed by Ron Rivest in 1991.

MD5 has 128-bit outputs. Wang and Yu (2004) found MD5 collisions in 2^{39} steps. MD5 collisions can

now be found in 2^{24} steps (in a few seconds on a laptop computer).

Sasaki & Aoki (2009) discovered a method for finding preimages for MD5 in $2^{123.4}$ steps.

Summary MD5 should not be used if collision resistance is required, but is probably okay as a preimage-resistant hash function.

MD5 is still used today. (2006) MD5 is implemented more than 850 times in Microsoft Windows source code. (2014) Microsoft issued a patch that restricts the use of certificates with MD5 in Windows: <http://tinyurl.com/MicrosoftMD5>

Flame Malware

See [https://en.wikipedia.org/wiki/Flame_\(malware\)](https://en.wikipedia.org/wiki/Flame_(malware))

Discovered in May 2012. Highly sophisticated espionage tool. Targeted computers in Iran and the Middle East. Suspected to originate from the US and/or Israeli government; US government has denied all responsibility. Contains a forged Microsoft certificate for Windows code signing. Forged certificate used a new, “zero-day MD5 chosen-prefix” collision attack. Microsoft no longer allows the use of MD5 for code signing.

3.5 SHA-1

Secure Hash Algorithm (SHA) was designed by NSA and published by NIST in 1993 (FIPS 180). 160-bit iterated hash function, based on MD4. Slightly modified to SHA-1 (FIPS 180-1) in 1994 in order to fix an (undisclosed) security weakness. Wang et al. (2005) found collisions for SHA in 239 steps. Wang et al. (2005) discovered a collision-finding algorithm for SHA-1 that takes 263 steps. The first SHA-1 collision was found on February 23, 2017. No preimage or 2nd preimage attacks (that are faster than the generic attacks) are known for SHA-1.

Microsoft’s SHA-1 Plan

See <http://tinyurl.com/MicrosoftSHA1>

As of May 9, 2017: TLS server-authentication certificates that use SHA-1 will be considered invalid.

Still unaffected:

- Code signature file hashes
- Code signing certificates
- Timestamp signature hashes
- Timestamping certificates
- etc.

3.6 SHA-2 Family

In 2001, NSA proposed variable output-length versions of SHA-1. Output lengths are 224 bits (SHA-224 and SHA-512/224), 256 bits (SHA-256 and SHA-512/256), 384 bits (SHA-384) and 512 bits (SHA-512). 2020: No weaknesses in any of these hash functions have been found. Note: The security levels of these hash functions against collision-finding attacks is the same as the security levels of Triple-DES, AES-128, AES-192 and AES-256 against exhaustive key search attacks. The SHA-2 hash functions are standardized in FIPS 180-2.

3.7 Description of SHA-256

Iterated hash function (Merkle meta-method).

$n = 256, r = 512$.

Compression function is $f : \{0, 1\}^{256+512} \rightarrow \{0, 1\}^{256}$.

Input: bitstring x of arbitrary bitlength $b \geq 0$.

Output: 256-bit hash value $H(x)$ of x .

SHA-256 Notation

A, B, C, D, E, F, G, H	32-bit words.
$+$	addition modulo 2^{32} .
\overline{A}	bitwise complement.
$A \gg s$	shift A right through s positions.
$A \hookrightarrow s$	rotate A right through s positions.
AB	bitwise AND.
$A \oplus B$	bitwise exclusive-OR.
$f(A, B, C)$	$AB \oplus \overline{AC}$.
$g(A, B, C)$	$AB \oplus AC \oplus BC$.
$r_1(A)$	$(A \hookrightarrow 2) \oplus (A \hookrightarrow 13) \oplus (A \hookrightarrow 22)$.
$r_2(A)$	$(A \hookrightarrow 6) \oplus (A \hookrightarrow 11) \oplus (A \hookrightarrow 25)$.
$r_3(A)$	$(A \hookrightarrow 7) \oplus (A \hookrightarrow 18) \oplus (A \gg 3)$.
$r_4(A)$	$(A \hookrightarrow 17) \oplus (A \hookrightarrow 19) \oplus (A \gg 10)$.

3.7.1 SHA-256 Constants

32-bit initial chaining values (IVs):

$$\begin{aligned}
 h_1 &= 0x6a09e667, \\
 h_2 &= 0xbb67ae85, \\
 h_3 &= 0x3c6ef372, \\
 h_4 &= 0xa54ff53a, \\
 h_5 &= 0x510e527f, \\
 h_6 &= 0x9b05688c, \\
 h_7 &= 0x1f83d9ab, \\
 h_8 &= 0x5be0cd19.
 \end{aligned}$$

These words were obtained by taking the first 32 bits of the fractional parts of the square roots of the first 8 prime numbers.

Per-round integer additive constants:

$$\begin{aligned}
 y_0 &= 0x428a2f98, \\
 y_1 &= 0x71374491, \\
 y_2 &= 0xb5c0fbcf, \\
 y_3 &= 0xe9b5dba5, \\
 &\vdots \\
 y_{62} &= 0xbef9a3f7, \\
 y_{63} &= 0xc67178f2.
 \end{aligned}$$

These words were obtained by taking the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers.

Algorithm 9: SHA-256 Preprocessing

```

1 Pad  $x$  (with 1 followed by as few 0's as possible) so that its bitlength is 64 less than a multiple
  of 512.
2 Append a 64-bit representation of  $b \bmod 2^{64}$ .
3 The formatted input is  $x_0, x_1, \dots, x_{16m-1}$ , where each  $x_i$  is a 32-bit word.
4 Initialize chaining variables:  $(H_1, H_2, \dots, H_6, H_7, H_8) \leftarrow (h_1, h_2, \dots, h_6, h_7, h_8)$ .
5 foreach  $i$  from 0 to  $m - 1$  do
6   Copy the  $i$ -th block of sixteen 32-bit words into temporary storage:  $X_j \leftarrow x_{16i+j}, 0 \leq j \leq 15$ .
   // Expand the 16-word block into a 64-word block:
7   for  $j$  from 16 to 63 do
8      $X_j \leftarrow r_4(X_{j-2}) + X_{j-7} + r_3(X_{j-15}) + X_{j-16}$ .
   // Initialize working variables:
9    $(A, B, \dots, F, G, H) \leftarrow (H_1, H_2, \dots, H_6, H_7, H_8)$ .
10  for  $j$  from 0 to 63 do
11     $T_1 \leftarrow H + r_2(E) + f(E, F, G) + y_j + X_j$ 
12     $T_2 \leftarrow r_1(A) + g(A, B, C)$ 
13     $H \leftarrow G, \quad G \leftarrow F, \quad F \leftarrow E, \quad E \leftarrow D + T_1$ 
14     $D \leftarrow C, \quad C \leftarrow B, \quad B \leftarrow A, \quad A \leftarrow T_1 + T_2$ .
15  Update chaining values:  $(H_1, H_2, \dots, H_7, H_8) \leftarrow (H_1 + A, H_2 + B, \dots, H_7 + G, H_8 + H)$ .
```

Output: $\text{SHA-256}(x) = H_1 \parallel H_2 \parallel H_3 \parallel H_4 \parallel H_5 \parallel H_5 \parallel H_6 \parallel H_7 \parallel H_8$.

Performance: Speed benchmarks (2017) for software implementations on an Intel Core i9 2.9 GHz 6-core Coffee Lake (8950HK) using OpenSSL 1.1.1d.

3.8 SHA-3

The SHA-2 design is similar to SHA-1, and thus there are lingering concerns that the SHA-1 weaknesses could eventually extend to SHA-2

SHA-3: NIST hash function competition. 64 candidates submitted by Oct 31 2008 deadline. 2012: Keccak was selected as the winner.

Keccak uses the “sponge construction” and not the Merkle iterated hash design. SHA-3 is being used in practice, but is not as widely deployed as SHA-2.

NIST’s Policy on Hash Functions

August 5, 2015

See: <http://csrc.nist.gov/groups/ST/hash/policy.html>

Should stop using SHA-1 for digital signatures and other applications that require collision resistance. May still use SHA-1 for HMAC, KDFs, and random number generators. May use SHA-2 for all applications that employ secure hash algorithms. SHA-3 may also be used, but this is not required.

4

Message authentication code schemes
