Introduction to Cryptography

F. Koeune – O. Pereira

MAT2450 – Lecture 5

Université catholique de Louvain
UCL
CARTE D'ACCÈS
Cette carte n'est ni une carte d'étudiant ni une carte de membre du personnel.
N° carte
N° FGS

Part I

Hash functions

Hash functions

Another most common cryptographic tool
Applications: MACs, password storage, key derivation, bitcoin, . . .

Idea: obtain some sort of “digest”, or “fingerprint” of a message

- Shorter than the message
- Easy to compute from it
- Uniquely identifying the message

Main requirement:

- It must be unfeasible to find two messages with the same hash
- Such a pair would be called a *collision*

More formally…

Difficult to deal with a *fixed* hash function

Instead, we will once again use a family of *keyed* hash functions

- H takes two inputs: a "key" s and a string x
- We define $H^s(x) := H(s, x)$

Important difference with previous schemes:

- Here s is *not* supposed to be secret!
- Idea: "anybody can compute the hash of a message"
- If s is not there: $\mathcal{A}$ can just have collisions "hardcoded"
  - $\forall \mathcal{A}$ includes $\mathcal{A}$ that knows collisions for 10^9 first output lengths

Collision-resistance:

- For a randomly-generated s, it must be hard to find x, x' such that $H^s(x) = H^s(x')$

Definition

A hash function is a pair $\Pi := \langle \mathrm{Gen}, H \rangle$

- Gen: given a security parameter 1^n, probabilistically selects a key s
- H takes as input a key s and a string x and outputs a string $H^s(x) \in \{0,1\}^{l(n)}$

H can be defined

- For fixed-length x: we have a fixed-length hash function
- For arbitrary-length x: arbitrary-length hash function

Security

We define the *collision-finding experiment* $\mathsf{HashColl}_{\mathcal{A},\Pi}(n)$ as

- Choose $s \leftarrow \mathrm{Gen}(1^n)$
- $\mathcal{A}$ is given s and outputs x, x' $\quad^{(*)}$
- Define $\mathsf{HashColl}_{\mathcal{A},\Pi}(n) := 1$ iff $H^s(x) = H^s(x')$

$^{(*)}$ With appropriate length restrictions if we consider a fixed-length scheme

Security

$\Pi := \langle \mathrm{Gen}, H \rangle$ is *collision-resistant* if $\forall$ PPT $\mathcal{A}$, $\exists\ \epsilon$:

$$\Pr[\mathsf{HashColl}_{\mathcal{A},\Pi}(n) = 1] \leq \epsilon(n)$$

Weaker security notions

Hash functions are typically required to have (informally):

1. *Collision resistance:* given s, difficult to find $x, x' : H^s(x) = H^s(x')$
2. *Second pre-image resistance:* given s, x, difficult to find $x' : H^s(x) = H^s(x')$
3. *Pre-image resistance:* given s and $y = H^s(x)$, difficult to find $x' : H^s(x') = y$

It is easy to see that $1 \Rightarrow 2 \Rightarrow 3$ when H is compressing

A generic attack against hash functions

Consider a hash function producing n-bit outputs

And suppose an adversary that tries to find a collision by simply trying inputs $x_1, \ldots, x_q$ at random

What are its chances of success?

A generic attack against hash functions

$\Rightarrow$ Birthday paradox

- Collision probability is about $\frac{q^2}{2.2^n}$

So, after $q = 2^{\frac{n}{2}}$, adversary has about 50% chances to win

This gives us a lower bound on practical hash output lengths

- If we want to avoid adversaries with computing power 2^n, we need $2n$-bit long hashes

Constructing hash functions

Constructing general-purpose hash functions is a very difficult task

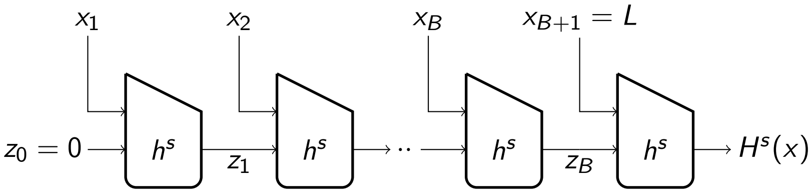
However, we can reduce the problem to that of constructing *fixed-size* hash functions

Most common solution:

- the Merkle-Damgård transform

The Merkle-Damgård transform

Suppose that we have a fixed-size hash function $\langle \mathrm{Gen}, h \rangle$
We can extend this to a hash function $\langle \mathrm{Gen}, H \rangle$ mapping arbitrary strings to l-bit strings as follows:



The Merkle-Damgård transform

For simplicity, assume h maps $2l$-bit strings to l-bit strings

- Gen: unchanged
- H: on input x of length $L < 2^{l(n)}$
  1. Parse x into $l(n)$-bit blocks $x_1, \ldots, x_B$
     (pad with 0s if necessary)
  2. Set $x_{B+1} := L$
  3. Set $z_0 := 0 \ldots 0$
  4. For $i = 1, \ldots, B+1$, compute $z_i := h^s(z_{i-1}||x_i)$
  5. Output z_{B+1}

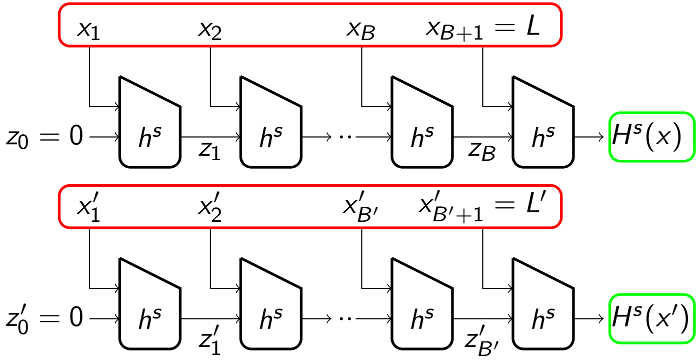
The Merkle-Damgård transform

Theorem: If $\langle \mathrm{Gen}, h\rangle$ is a fixed-length collision-resistant hash function, then $\langle \mathrm{Gen}, H\rangle$ is a collision-resistant hash function

Proof: We will show that any collision on H implies a collision on h

Suppose we have a collision (for H^s)

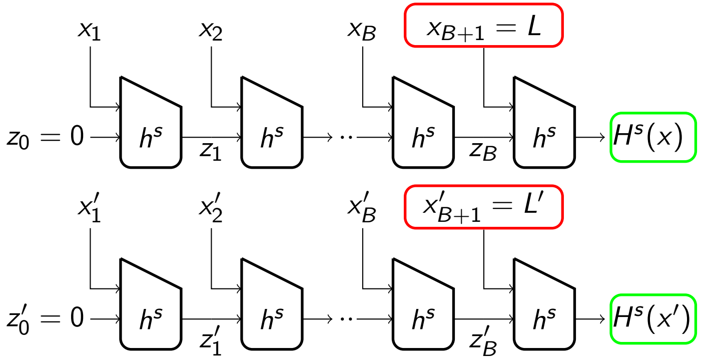
So, we have two different sequences. . .



. . . yielding the same hash

Case 1: different lengths

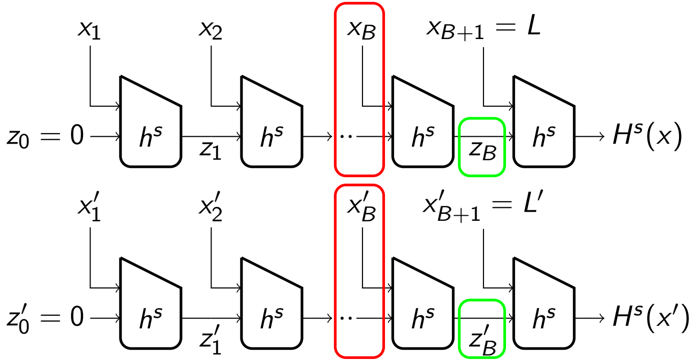
If lengths differ...



...then this is a collision (for h^s)

Case 2: same lengths

Consider the last place where these pairs differ. . .



. . . by assumption, these values are equal

. . . so this is a collision (for h^s)

Proof (sketch)

Suppose we have two strings x, x' (of length L, L'):
$H^s(x) = H^s(x')$

If $L \neq L'$

- Then $H^s(x) = z_{B+1} = h^s(z_B||L)$
- And $H^s(x') = z'_{B+1} = h^s(z'_B||L')$
- As $H^s(x') = H^s(x')$ and $L \neq L'$, this is a h^s collision

Proof (sketch)

If $L = L'$

- As $x \neq x', \exists i : x_i \neq x'_i$
- Let i^* be the highest i for which $z_{i^*-1}||x_{i^*} \neq z'_{i^*-1}||x'_{i^*}$
- If $i^* = B+1$
  - Then $H^s(x) = z_{B+1} = h^s(z_B||x^{B+1})$
  - And $H^s(x') = z'_{B+1} = h^s(z'_B||x'^{B+1})$

  $\Rightarrow$ This is a collision
- If $i^* \leq B$
  - Then $z_{i^*} = z'_{i^*}$ (i^* max)
  - So $h^s(z_{i^*-1}||x_{i^*}) = h^s(z'_{i^*-1}||x'_{i^*})$

  $\Rightarrow$ This is a collision

Full proof

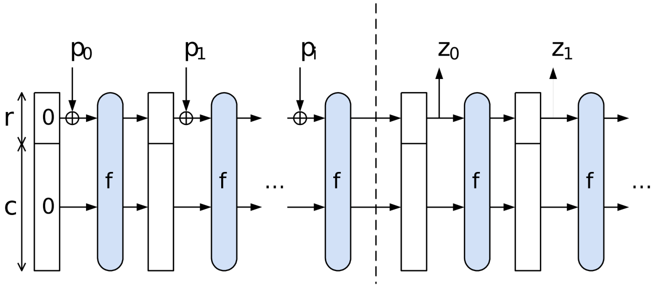
For the proof to be complete, we would need

- to consider an adversary $\mathcal{A}$ that can break H
- to use $\mathcal{A}$ in order to build an adversary $\mathcal{A}'$ that can break h

$\Rightarrow$ Quite straightforward based on the proof sketch (try it!)

Another Approach

The sponge construction [Bertoni et al., 2006]:



- *f* is a permutation (or transformation): no compression
- *r* is the bit-rate, *c* the capacity
- output produced through many "squeezes"
- output can have arbitrary length

Practical constructions of hash functions

Generally unkeyed ($\Rightarrow$ one function, not a family)

- Practical $\mathcal{A}$ needs to find one collision
- Much more damaging if some actual collision is found
- Yet, we can hope to achieve "practical" security

Security:

- Collision should cost birthday bound
- $H(x)$ should look random to anyone who did not compute it from x

Actual hash function examples

MD5 [Rivest, 1992]

- Produces 128-bit output
  - Birthday paradox: can be attacked with 2^{64} complexity
- Significant weakness found in 2004
  - Today, collisions produced in < 1 sec
- But still used:
  - Flame cyber-weapon [2012] authenticated itself as Microsoft validated thanks to MD5 legacy use
  - Yahoo 1B account breach [2013] showed MD5 protected passwords

Practical applications of the attack against MD5

- `www.cits.rub.de/MD5Collisions/`: two postscript files with the same MD5 hash
- `www.codeproject.com/dotnet/HackingMd5.asp`: two pieces of code with same MD5 hash
- `www.win.tue.nl/~bdeweger/CollidingCertificates/`: colliding X.509 Certificates based on MD5 collisions

Actual hash function examples

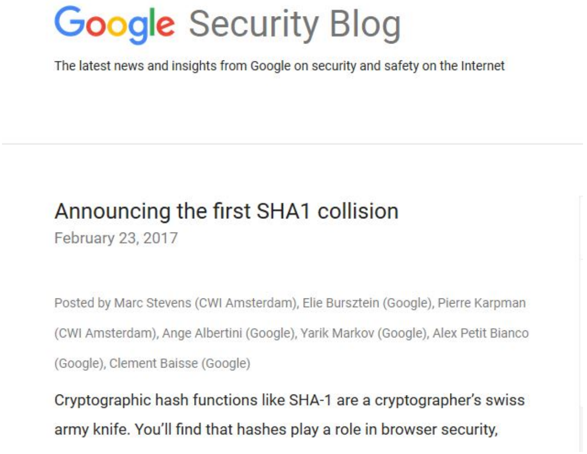
SHA-1 [NSA, 1995]

- Produces 160-bit output
- Attacks improving since 2005
  - Collision found in Feb. 2017
- Phasing out: browsers stop supporting it in 2017

SHA-2 [NSA, 2001]

- A family of successors to SHA-1
- SHA-256, SHA-384, SHA-512
- No significant weakness known, most common today

Collision on SHA-1



Google Security Blog

The latest news and insights from Google on security and safety on the Internet

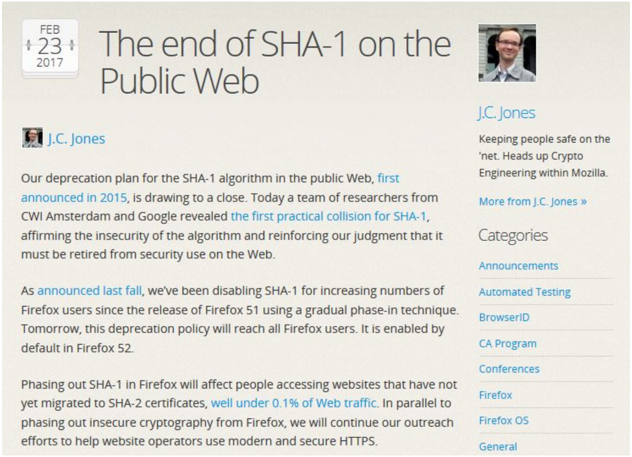
Announcing the first SHA1 collision

February 23, 2017

Posted by Marc Stevens (CWI Amsterdam), Elie Bursztein (Google), Pierre Karpman (CWI Amsterdam), Ange Albertini (Google), Yarik Markov (Google), Alex Petit Bianco (Google), Clement Baisse (Google)

Cryptographic hash functions like SHA-1 are a cryptographer's swiss army knife. You'll find that hashes play a role in browser security,

Actual hash function examples



FEB 23 2017

The end of SHA-1 on the Public Web

J.C. Jones

Our deprecation plan for the SHA-1 algorithm in the public Web, first announced in 2015, is drawing to a close. Today a team of researchers from CWI Amsterdam and Google revealed the first practical collision for SHA-1, affirming the insecurity of the algorithm and reinforcing our judgment that it must be retired from security use on the Web.

As announced last fall, we've been disabling SHA-1 for increasing numbers of Firefox users since the release of Firefox 51 using a gradual phase-in technique. Tomorrow, this deprecation policy will reach all Firefox users. It is enabled by default in Firefox 52.

Phasing out SHA-1 in Firefox will affect people accessing websites that have not yet migrated to SHA-2 certificates, well under 0.1% of Web traffic. In parallel to phasing out insecure cryptography from Firefox, we will continue our outreach efforts to help website operators use modern and secure HTTPS.

J.C. Jones

Keeping people safe on the 'net. Heads up Crypto Engineering within Mozilla.

More from J.C. Jones »

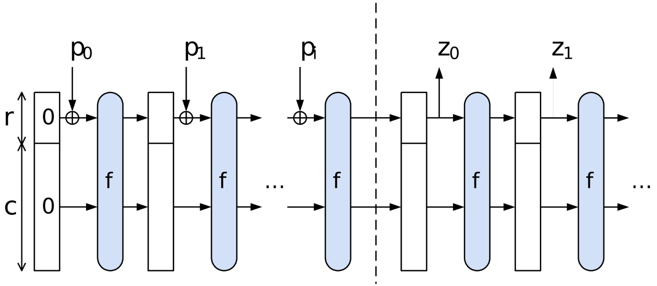
Categories

- Announcements
- Automated Testing
- BrowserID
- CA Program
- Conferences
- Firefox
- Firefox OS
- General

Actual hash function examples

SHA-3 [Bertoni et al., 2008]

- NIST standard since 2015 (after 5 years of competition)
- Slower adoption
- Sponge construction



MAC construction revisited

Hash functions can help build efficient MAC schemes

Idea: hash something depending on

- the message
- and a secret key

e.g. $t := H^s(k||m)$

Intuition:

- small modifications in message completely change its hash

$\Rightarrow$ without knowing the key, nothing can be predicted on tag

Building a MAC with a hash function

Simple constructions (e.g. $H^s(k||m)$) were first proposed, but bear flaws

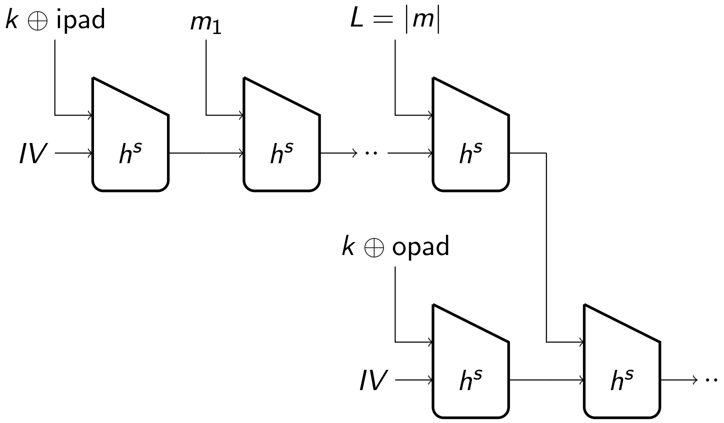
A secure construction: HMAC [Bellare et al. 1996]

$$t := H^s((k \oplus \mathrm{opad})||H^s((k \oplus \mathrm{ipad})||m))$$

where $(\mathrm{opad}, \mathrm{ipad})$ are fixed constants

Can be proved secure under some assumptions on the underlying h

HMAC with Merkle-Damgård



Password storage

Motivation:

- Server that needs to verify pw's for millions of users

0. Never store passwords in clear!
1. Store $(login, H(pw))$ and recompute $H(pw)$ when user comes?
   Hacking server does not leak stored passwords
   But brute force search from common passwords is quite easy:
   - Computing $H(x)$ for $x \in \{$123456, password, 12345678, qwerty, 123456789, 12345, 1234, 111111, 1234567, dragon$\}$ already makes you spot 1.6% of the passwords

[WP Engine]

Password storage

Motivation:

- Server that needs to verify pw's for millions of users

2. Store $(login, salt, H(salt \| pw))$ for long random salt?
   Brute force search is now one password at a time!
3. Replace $H(\cdot)$ with $H(H(H(\cdots H(\cdot)\cdots)))$
   Slows down exhaustive search (Bcrypt, PBKDF2, ...)
   But specific hardware can still bring considerable speedup
4. Tweak hash iterations to use bits from past iterations in an upredictable way
   Takes memory, makes parallelism difficult
   (scrypt, Argon2, ...)

Create tamper-evident logs

Maintain hash chain:

- Start with $h_0 = init$
- Define $h_i = H(h_{i-1}, t_i, event_i)$
- Append $\langle h_i, t_i, event_i \rangle$ to log

Any log tampering requires recomputing the whole chain

h_i can be replicated, . . .

Part II

Practical constructions of block ciphers

Practical constructions of block ciphers

Objectives

- Review some of the key principles of today's techniques to build pseudorandom permutations (i.e. block ciphers) *candidates*
  - Substitution-permutation network
  - Feistel scheme
  - ...
- Get a global idea of some of the most popular block ciphers today
- Get an idea of *practical* security bounds: which key size is deemed secure today?
- Highlight some issues when trying to improve security

Basic building blocks

So far, we have

- Defined the notion of security
- Developed constructions that achieved that security *provided* we have access to a strong building block (PRG, PRF, PRP...)

Now we will try to build these basic blocks

Unfortunately, these constructions will most of the time be heuristic

Heuristic vs. provable security

Provable security:

- "We have a formal proof that there cannot exist an adversary capable of performing a specific operation" (decrypt, forge MAC, . . .)
- Note: proof is often conditional: ". . . *provided* this well-known problem is difficult"

$\Rightarrow$ No assumption at all about *how* the adversary is supposed to proceed

Heuristic vs. provable security

Heuristic security

- "We have tried very hard to break the construction and could not find a way of breaking it"
- Involves strong cryptanalysis effort: try all well-known attack techniques and give convincing arguments that they will not work

$\Rightarrow$ But we could have missed a completely new adversary that can break the construction

Why using heuristic schemes?

Because we do not really have choice!

Today, we do not have efficient and provably secure building blocks (but research continues)

If we want to have usable cryptography (i.e. that will not consume the main part of the resources), we must roll back to heuristic constructions

Can we trust these schemes?

Although we have no proof, current standard block ciphers are based on well-studied techniques that have withstood intensive analysis

Examples:

- Data Encryption Standard (DES) [1976] has no practical weakness so far (parameters just became too small)
- Advanced Encryption Standard (AES) [1998], no known attack, most widespread today

Both are based on generic techniques that have also been extensively scrutinized (and for which some "partial proofs" also exist)

The global approach

- Try to limit the heuristic aspect as much as possible
- Use heuristic basic blocks, combined through provably secure constructions
  - We can limit the uncertainty to a very specific subpart and define exactly what we expect from it
  - If block gets broken, just replace it by another one, and keep the construction
- Also the approach taken in this course

Concrete vs. asymptotic security

So far, we considered our constructions as functions of a security parameter

- e.g. $F : \{0,1\}^n \times \{0,1\}^l \rightarrow \{0,1\}^l$

And showed that security was asymptotically reached

- i.e. $\mathcal{A}$'s chances of success become negligible as n grows

Now we will consider *fixed constants* for n and l

- We are thus dealing with concrete security: "is this impossible for an adversary *today*?"
- Yet, we will still use parameters size as a comparison factor between different block ciphers
  - But we must keep in mind that longer keys are a *necessary* but not *sufficient* condition for additional security
  - This comparison is thus heuristic too

Usefulness of PRPs

Let F be a PRP: $\{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$

Then F is also a PRF

Proof idea:

- PRF attacker queries relayed to and from PRP challenger
- Only way to distinguish is that PRFs have collisions
- But we can only hope to observe collisions after $2^{n/2}$ queries (birthday paradox)

So, enough to focus on building good PRPs, and then use them as PRFs when needed.

Usefulness of PRPs

Let F be a PRP: $\{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$.
Then $G : s \mapsto F_s(0) \| F_s(1)$ is a PRG.

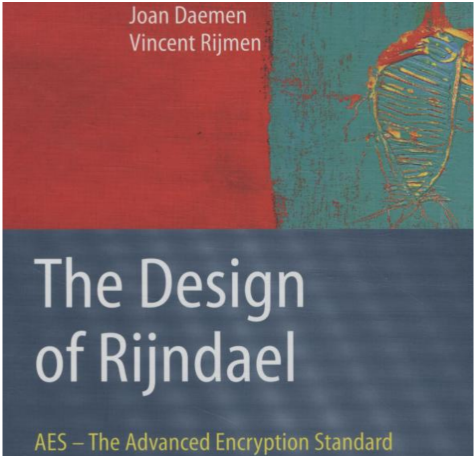
So, a PRP also gives us a PRG!

$\Rightarrow$ We will focus on block ciphers:

- Other name for PRP
- Typically defined for just a few values of n

The Design of Rijndael

250+ pages on the design of the AES Rijndael:


Joan Daemen
Vincent Rijmen

The Design of Rijndael

AES – The Advanced Encryption Standard

Definition of a breaking

Practical requirement is:

Distinguishing from random permutation should be roughly as hard as exhaustive key search.

- So, $2^{n/2}$ security would not be enough, even if not PPT
- Motivations:
  - better than brute-force is usually a bad sign
  - keeps parameters as small as possible ($n = 128$ today)

Substitution-permutation networks

A block cipher is a pseudorandom permutation

Problem: describing a full n bits permutation requires $\approx n.2^n$ bits

- Practical block sizes today: 128 bits
- Impossible to achieve

Idea: let us try to build one from smaller permutations

Substitution-permutation networks

Suppose we have 16 (key dependent) 8-bit random permutations

$$f_1(\cdot), \dots f_{16}(\cdot)$$

Can we define $F(x) := f_1(x_1) || \dots || f_{16}(x_{16})$?

Quizz: consider x, x' differing by only 1 bit. What will $F(x)$ and $F(x')$ look like?

1. Differ by 1 bit
2. Differ by a few bits
3. Be totally different

And what should they look like for F to be a PRP?

Substitution-permutation networks

Concatenating small random permutations does not work

- If x and x' differ by 1 bit, $F(x)$ and $F(x')$ will differ by at most 8 bits
- This is certainly not pseudorandom

However, if we

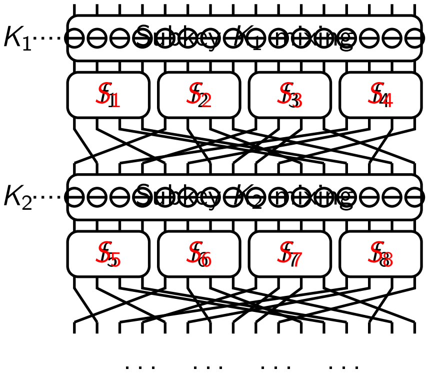
- Permute the bits of x after applying F
- Repeat these two steps several times

Then we can hope to achieve something close to a PRP

This is also known as the *confusion-diffusion paradigm* [Shannon, 1945]

- Substitution $\approx$ Caesar's cipher
- Permutation $\approx$ Scytale cipher

Substitution-permutation networks



A SPN is an application of the confusion-diffusion paradigm,

- Ideally, new f_i picked for each use
- But difficult in practice
- Instead, we use fixed S-boxes
- And key-dependence achieved by combining key bits with input to S-boxes
  - Different parts of key used at each round, according to a *key schedule*
- Key combination often simply $\oplus$

Beware, do not mix up

- The word “permutation” is used with 2 different meanings in cryptography
  - A pseudorandom permutation is a (pseudorandom) function that is one-to-one (i.e. injective and surjective)
  - In a SPN, a permutation is a reordering of the bits
- Keep in mind that a PRP does not simply reorder bits !

Basic design principles

1. S-boxes must be invertible
   - Otherwise we would not get a permutation
   - Note that this is a sufficient condition for full invertibility
2. Avalanche effect: each change (even local) to the input must result in a large change in the output. For this:
   - Changing a single bit in a S-box input should change at least two bits in the output
   - The mixing permutation should ensure that the output bits of a given S-box are spread into different S-boxes in the next round

Basic design principles

S-boxes, mixing permutations, key schedule and number of rounds are what makes the difference between a strong and a weak block cipher

Involves a very careful analysis, taking into account many properties and known attack techniques

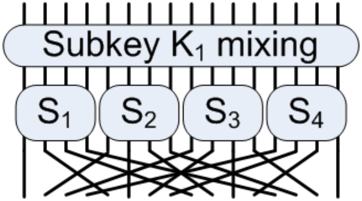
$\Rightarrow$ do not try building your own block cipher

Attacking reduced-round versions of a SPN

To get a better insight in the substitution-permutation paradigm, let us try attacking reduced versions of a SPN, with fewer rounds

As before, we consider an adversary who receives input-output pairs and must tell whether he is facing a random permutation or a SPN

One-round SPN

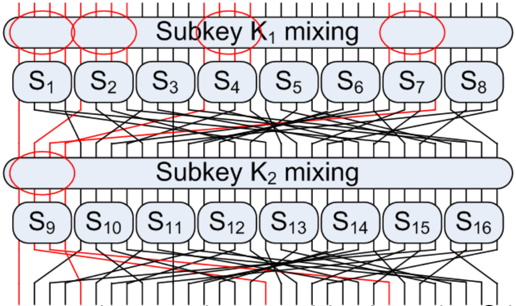


With just one input-output pair, $\mathcal{A}$ can

- Undo mixing on output (public design)
- Undo S-boxes (public design)
- XOR with input and recover the key

$\Rightarrow$ Can trivially distinguish the SPN from a RF

Two-round SPN



These four output bits can be traced back to this S-box and thus depend on these 16 input bits, 4 bits of K_2 and 16 bits of K_1

$\Rightarrow$ Exhaustive search on this partial key is possible

$\Rightarrow$ And thus can of course also be distinguished from a RF

Three-round SPN

Avalanche effect is not complete after three rounds, $\mathcal{A}$ can

- Send strings differing in only one bit
- Observe whether outputs are affected locally or globally

$\Rightarrow$ This will allow telling the difference between SPN and a PRF

Remark: As a comparison, the Advanced Encryption Standard (AES-128) has 10 rounds

Advanced Encryption Standard (AES)

NIST's standardization project

- First call in 1997 (15 candidates)
- Final decision in 2000

Final decision: Rijndael

- 128-bit block cipher
- Substitution-permutation network
- 3 key sizes: 128, 192 or 256 bits

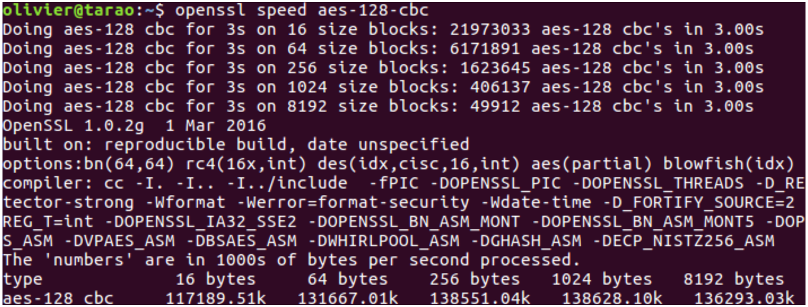
The AES in practice

Adoption: almost immediate and ubiquitous

- Network: TLS, SSH, . . .
- Disk encryption (BitLocker, LibreCrypt, TrueCrypt, . . .)
- Archive and compression tools (7z, RAR, WinZip, KeePass, . . .)
- Implementations available for “all” languages

The AES in practice

Speed $> 100\text{MB/s}$ on single core



```
olivier@tarao:~$ openssl speed aes-128-cbc
Doing aes-128 cbc for 3s on 16 size blocks: 21973033 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 64 size blocks: 6171891 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 256 size blocks: 1623645 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 1024 size blocks: 406137 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 8192 size blocks: 49912 aes-128 cbc's in 3.00s
OpenSSL 1.0.2g  1 Mar 2016
built on: reproducible build, date unspecified
options:bn(64,64) rc4(16x,int) des(idx,cisc,16,int) aes(partial) blowfish(idx)
compiler: cc -I. -I.. -I../include  -fPIC -DOPENSSL_PIC -DOPENSSL_THREADS -D_RE
tector-strong -Wformat -Werror=format-security -Wdate-time -D_FORTIFY_SOURCE=2
REG_T=int -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOP
S_ASM -DVPAES_ASM -DBSAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM -DECP_NISTZ256_ASM
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128 cbc     117189.51k   131667.01k   138551.04k   138628.10k   136293.03k
```

AES-NI instruction set on processors $\Rightarrow \approx 8\times$ speedup

So far, no known attack better than exhaustive search

Feistel networks

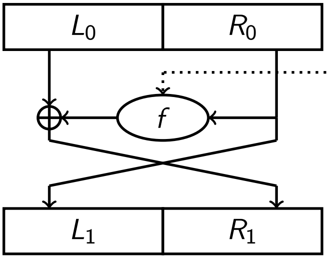
Suppose that we have a good PRF, yet not invertible (i.e. not a permutation)

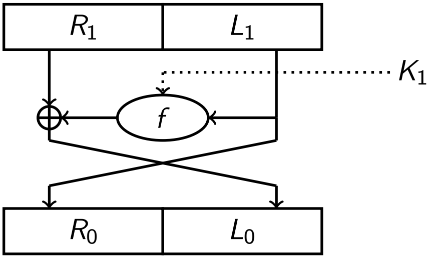
Can we build a block cipher from this?

Yes

- One way of achieving this has been proposed by Feistel

Feistel networks





Encryption

- $L_1 := R_0$
- $R_1 := f_{K_1}(R_0) \oplus L_0$

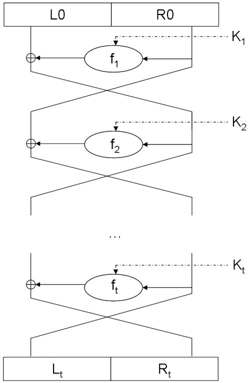
Decryption

- $R_0' := L_1$
- $L_0' := f_{K_1}(L_1) \oplus R_1$

Of course, this is not a good encryption scheme

- But if we iterate it...

Feistel networks

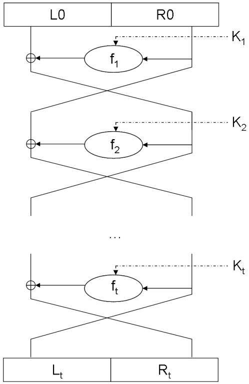


It can be proved (Luby-Rackoff) that

If *f* is a PRF,

- Then a 3-round Feistel network is a PRP
- And a 4-round Feistel network is a *strong* PRP (i.e. strong even if distinguisher is given oracle access to the inverse of the function)

In practice...



- f_i functions are constructed in a similar way as for SPN
- f_i are typically fixed, with key dependence ensured by combining ($\oplus, \ldots$) input with subkeys
- Subkeys are derived according to some key schedule

Feistel networks

Advantages

- More latitude in the choice of f_i
- Same software/hardware can be used for encryption and decryption (just revert the key schedule)

Feistel networks adoption:

- DES [1975–1999–2005] (IBM, NSA)
- Camellia [2000—] (Mitsubishi, NTT)
- ...

Data Encryption Standard

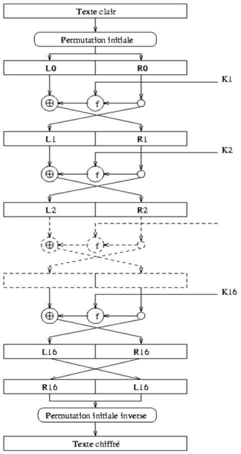
- US encryption standard (1977) Designed by IBM / NSA
- Feistel Network
- 64-bit block cipher
- 56-bit keys

These sizes are to short for today's computing power

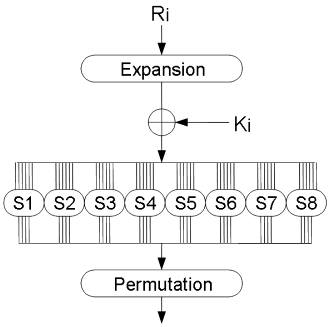
- DES should not be used any more
- Yet, still present in many "legacy" applications
- Some attacks found, but best practical attack known so far is exhaustive search

$\Rightarrow$ "A remarkable success story in cryptography"

Data Encryption Standard



The f function



Essentially, a 1-round SPN

- Expansion: 32 bits $\rightarrow$ 48 bits (by duplication)
- S-boxes: substitutions 6 bits $\rightarrow$ 4 bits
- Permutation: reorders 32-bit output

DES security

In 1990, a new class of attacks, *differential cryptanalysis*, was invented by Biham and Shamir

- Attack requires 2^{47} chosen plaintexts and has 2^{37} complexity
- Remark: it turned out that the DES S-boxes were designed to resist that (at that time publicly unknown) attack

In 1993, another new class, *linear cryptanalysis*, was invented by Matsui

- Attack requires 2^{43} known plaintext

Yet, the best practical attack against DES is *brute-force key search*: complexity 2^{55}

- Remark: why not 2^{56}?

DES key size

Moore's law: computing power doubles every 18 months

$\Rightarrow$ Exhautive search on DES key becomes possible

- Dedicated machines
- Distributed effort
- To measure this, DES challenges have been introduced
  - 1997: March 13th $\rightarrow$ June 17th (internet)
  - January 1998: 39 days (internet)
  - July 1998: 56 hours (dedicated machine)
  - 1999: 22 hours (dedicated machine + 100 000 PCs $\approx$ \$250k)
  - 2006: $\approx$ 1 day (Copacabana: dedicated FPGAs $\approx$ \$10k)

Triple-DES

Double encryption adds little security

Use triple encryption

- With 3 keys
- Or just with 2
- Classical way: EDE mode
  - $c = \mathrm{Enc}_{K_1}(\mathrm{Dec}_{K_2}(\mathrm{Enc}_{K_1}(m)))$

But does not solve small block size problem

- (remember birthday paradox discussion at lecture 3)

Still used in some legacy applications

Conclusion

Block ciphers:

- One of the most versatile crypto objects:
  PRG, encryption, authentication, . . .
- Most common: AES, 128-bit blocks, 128-192-256-bit keys
  Many countries have their own BC for “internal” use
- AES implementations widely available,
  often with HW support
  Think twice before picking something else
  (Efficiency, side-channel resistance, . . .)