5. Other Cryptographic Constructions Relying on Coding Theory

- Code-Based Digital Signatures
- The Courtois-Finiasz-Sendrier (CFS) Construction
- Attacks against the CFS Scheme
- Parallel-CFS
- Stern's Zero-Knowledge Identification Scheme
- An Efficient Provably Secure One-Way Function
- **The Fast Syndrome-Based (FSB) Hash Function**

Requirements for a Cryptographic Hash Function

A cryptographic hash function has the following properties:

- its input can be of arbitrary size
- its output is a hash of fixed size
- from a security point of view, it should be hard to:
  - find an input with a given hash (preimage attack)
  - find an input with the same hash as a given input (second preimage)
  - find two inputs with the same hash (collision attack)

Requirements for a Cryptographic Hash Function

A cryptographic hash function has the following properties:

- its input can be of arbitrary size
- its output is a hash of fixed size
- from a security point of view, it should be hard to:
  - find an input with a given hash (preimage attack)
  - find an input with the same hash as a given input (second preimage)
  - find two inputs with the same hash (collision attack)

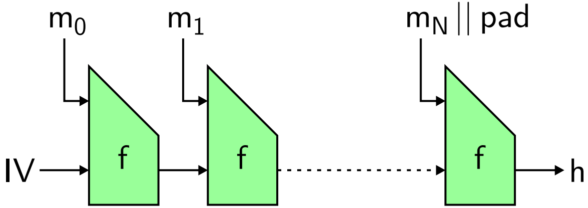
In addition, it should, as much as possible:

- be fast in both software and hardware implementations
- be fast for both small and large inputs
- have a compact description

Building a Cryptographic Hash Function

Building a function with arbitrary input length is tricky
→ usually, iterate a function with fixed input size on blocks of the input

The Merkle-Damgård Construction



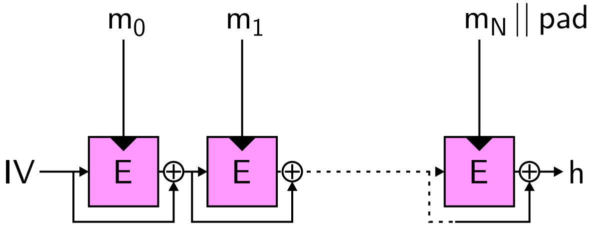
One of the first hash function constructions:

- *f* is a compression function
- easy to understand, simple security proofs

Building a Cryptographic Hash Function

Building a function with arbitrary input length is tricky
→ usually, iterate a function with fixed input size on blocks of the input

The Davies-Meyer Construction



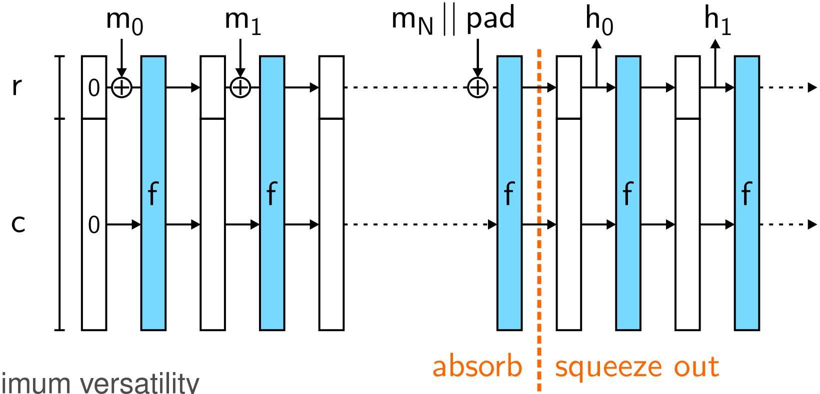
Ideal for compact implementations:

- *E* is a block cipher
- can reuse the same hardware

Building a Cryptographic Hash Function

Building a function with arbitrary input length is tricky
→ usually, iterate a function with fixed input size on blocks of the input

The Sponge Construction



- Maximum versatility

Overview of the Fast Syndrome-Based Hash Function

Uses the Merkle-Damgård construction.

Allows simple security analysis:

- properties of the compression function are transferred to the hash function
  - preimage resistance
  - second preimage resistance
  - collision resistance

  → analyse only the compression function

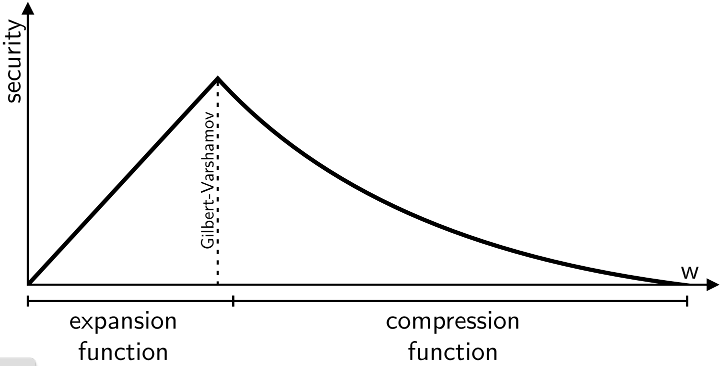
- has some drawbacks, but not so problematic:

  → long message collisions, multi-collisions...

Overview of the Fast Syndrome-Based Hash Function

Uses the Merkle-Damgård construction.

Uses the one-way function (previous session) with compression parameters.

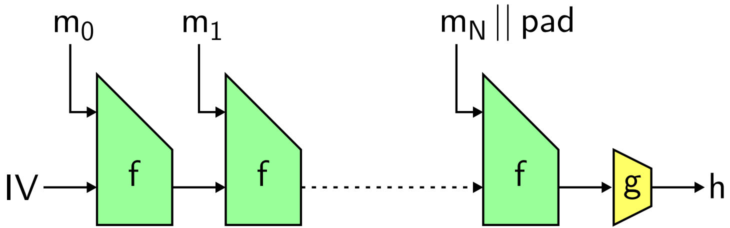


Overview of the Fast Syndrome-Based Hash Function

Uses the Merkle-Damgård construction.

Uses the one-way function (previous session) with compression parameters.

Adds a final compression function.



Description of FSB_{256}

Compression function:

- the matrix H is of size $r = 1024$ by $n = 2^{21}$
- the input of $s = 1792$ bits is encoded into a regular word of weight $w = 128$
  → each position is coded on $\frac{s}{w} = \log \frac{n}{w} = 14$ bits
- the output of $r = 1024$ bits is the XOR of 128 columns of H

Description of FSB_{256}

Compression function:

- the matrix H is of size $r = 1024$ by $n = 2^{21}$
- the input of $s = 1792$ bits is encoded into a regular word of weight $w = 128$
  → each position is coded on $\frac{s}{w} = \log \frac{n}{w} = 14$ bits
- the output of $r = 1024$ bits is the XOR of 128 columns of H

Chaining:

- the message to hash is split in blocks of $s - r = 768$ bits
- a padding is added to get an integer number of blocks
  → includes the message length
- the IV is all 0
- the compression function is iterated on the blocks
- the final output of $r = 1024$ bits is input to Whirlpool
  → the final hash has 256 bits

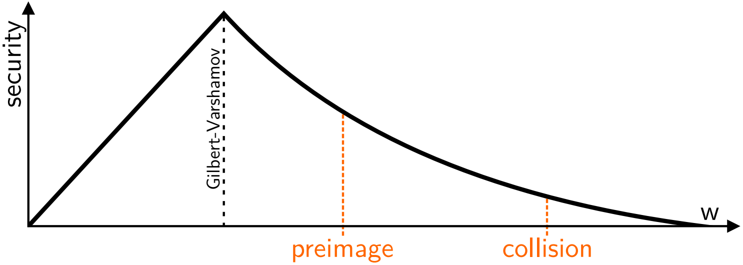
Security of the Compression Function

Against (second) preimage:

- solve a regular instance of SD with weight 128 and a 1024×2^{21} matrix
- best attack: GBA with complexity $2^{261} > 2^{256}$

Against collision:

- solve a regular instance of SD with weight 256 and a 1024×2^{21} matrix
- best attack: ISD with complexity $2^{153} > 2^{128}$



Need for a Final Compression Function

A hash function is expected to have the following properties:

- a security of $2^{\frac{r}{2}}$ against collisions for an output of r bits
  → this is the cost of a generic attack using a birthday algorithm
- it should be possible to truncate the output without losing security

Need for a Final Compression Function

A hash function is expected to have the following properties:

- a security of $2^{\frac{r}{2}}$ against collisions for an output of r bits
  → this is the cost of a generic attack using a birthday algorithm
- it should be possible to truncate the output without losing security

This is not the case for the compression function of FSB:

- if w allows compression, GBA with 4 lists is always possible on weight $2w$
  → the security is at most $2^{\frac{r}{3}}$ against collisions
- truncating the output directly improves GBA/ISD attacks
  → this is not desirable

Need for a Final Compression Function

A hash function is expected to have the following properties:

- a security of $2^{\frac{r}{2}}$ against collisions for an output of r bits
  → this is the cost of a generic attack using a birthday algorithm
- it should be possible to truncate the output without losing security

This is not the case for the compression function of FSB:

- if w allows compression, GBA with 4 lists is always possible on weight $2w$
  → the security is at most $2^{\frac{r}{3}}$ against collisions
- truncating the output directly improves GBA/ISD attacks
  → this is not desirable

Simply add a final compression:

- must be non-linear
- does not have to be collision/preimage resistant

Efficiency

Hashing speed:

- each 14 bits of input add a 1024 bit XOR
  → theoretically, could be as low as 10 cycles per byte (a 64 bit XOR per cycle)
- in practice, requires 300 cycles per input byte
  → around 10 MB/s hashing

Efficiency

Hashing speed:

- each 14 bits of input add a 1024 bit XOR
  → theoretically, could be as low as 10 cycles per byte (a 64 bit XOR per cycle)
- in practice, requires 300 cycles per input byte
  → around 10 MB/s hashing

Size of the description:

- H has a size of $2^{10} \times 2^{21}$ bits, that is 256 MB
  → this is way too much!
- instead a quasi-cyclic matrix is used
  → each 1024×1024 block is circulant
- only the first line of the matrix is needed
  → the description is 1024 times smaller: 256 kB

5. Other Cryptographic Constructions Relying on Coding Theory

We have seen several constructions relying on the hardness of Syndrome Decoding:

- McEliece, Niederreiter
- the CFS signature
- Stern's identification scheme
- the FSB hash function

Many other applications of coding theory in cryptography:

- secret sharing
- linear diffusion in block ciphers
- fingerprinting and traitor tracing
- private information retrieval