Department of
Computer Science and Engineering
计算机科学与工程系

CSE5014 CRYPTOGRAPHY AND NETWORK SECURITY

Dr. Qi Wang

Department of Computer Science and Engineering
Office: Room413, CoE South Tower
Email: wangqi@sustech.edu.cn

Constructions of Authenticated Encryption

- There are three natural generic constructions:
  - Encrypt and Authenticate (E&A): Compute $c = Enc_k(m)$ and $t = Mac_{k_2}(m)$ and send (c, t) (SSH style)
  - Authenticate and then Encrypt (AtE): Compute $t = Mac_{k_2}(m)$ and then $Enc_{k_1}(t)$ (SSL style)
  - Encrypt and then Authentication (EtA): Compute $c = Enc_{k_1}(m)$ and $t = Mac_{k_2}(c)$ and send (c, t) (IPSec style)

Note: In all these methods, we use independent keys for encryption and authentication

Hash functions

- (Cryptographic) *hash function*: deterministic function mapping arbitrary length inputs to a short, fixed-length output (sometimes called a *digest*)

Hash functions

- (Cryptographic) *hash function*: deterministic function mapping arbitrary length inputs to a short, fixed-length output (sometimes called a *digest*)

- Hash functions can be *keyed* or *unkeyed*
  - In practice, hash functions are unkeyed
  - We will assume unkeyed hash functions for simplicity

Collision-resistence

- Let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a *hash function*
- A *collision* is a pair of distinct inputs x, x' such that $H(x) = H(x')$.

Collision-resistence

- Let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a *hash function*
- A *collision* is a pair of distinct inputs x, x' such that $H(x) = H(x')$.
- **Theorem 7.1** H is *collision-resistant* if it is infeasible to find a collision in H.

Collision-resistence

- Let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a *hash function*
- A *collision* is a pair of distinct inputs x, x' such that $H(x) = H(x')$.
- **Theorem 7.1** H is *collision-resistant* if it is infeasible to find a collision in H.
- What is the best "generic" collision attack on a hash function $H; \{0,1\}^* \to \{0,1\}^\ell$?
  - Note that collisions are guaranteed to exist!

Collision-resistence

- Let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a *hash function*
- A *collision* is a pair of distinct inputs x, x' such that $H(x) = H(x')$.
- **Theorem 7.1** H is *collision-resistant* if it is infeasible to find a collision in H.
- What is the best "generic" collision attack on a hash function $H; \{0,1\}^* \to \{0,1\}^\ell$?
  - Note that collisions are guaranteed to exist!
  - If we compute $H(x_1), \ldots, H(x_{2^\ell+1})$, we are guaranteed to find a collision.
  - Can we do better?

"Birthday" attacks

- Compute $H(x_1), \ldots, H(x_k)$
  - What is the probability of a collision?

"Birthday" attacks

- Compute $H(x_1), \ldots, H(x_k)$
  - What is the probability of a collision?

- Related to the so-called *birthday paradox*
  - How many people are needed to have a 50% chance that some two people share a birthday?

"Birthday" attacks

- Compute $H(x_1), \ldots, H(x_k)$
  – What is the probability of a collision?

- Related to the so-called *birthday paradox*
  – How many people are needed to have a 50% chance that some two people share a birthday?

Let $n(p; H)$ be the smallest number of values we have to choose, such that the probability for finding a collision is at least p. By inverting the expression above, we have

$$n(p; H) \approx \sqrt{2H \ln \frac{1}{1-p}}.$$

“Birthday” attacks

- Compute $H(x_1), \ldots, H(x_k)$
  – What is the probability of a collision?

- Related to the so-called *birthday paradox*
  – How many people are needed to have a 50% chance that some two people share a birthday?

- When $k \approx H^{1/2}$, probability of a collision is $\approx 50\%$
  – Birthdays: 23 people suffice!
  – Hash functions: $O(2^{\ell/2})$ hash-function evaluations

"Birthday" attacks

- Compute $H(x_1), \ldots, H(x_k)$
  - What is the probability of a collision?

- Related to the so-called *birthday paradox*
  - How many people are needed to have a 50% chance that some two people share a birthday?

- When $k \approx H^{1/2}$, probability of a collision is $\approx 50\%$
  - Birthdays: 23 people suffice!
  - Hash functions: $O(2^{\ell/2})$ hash-function evaluations

- Need $\ell = 2n$-bit output length to get security against attackers running in time 2^n

"Birthday bound"

- The *birthday bound* comes up in many other cryptographic contexts

"Birthday bound"

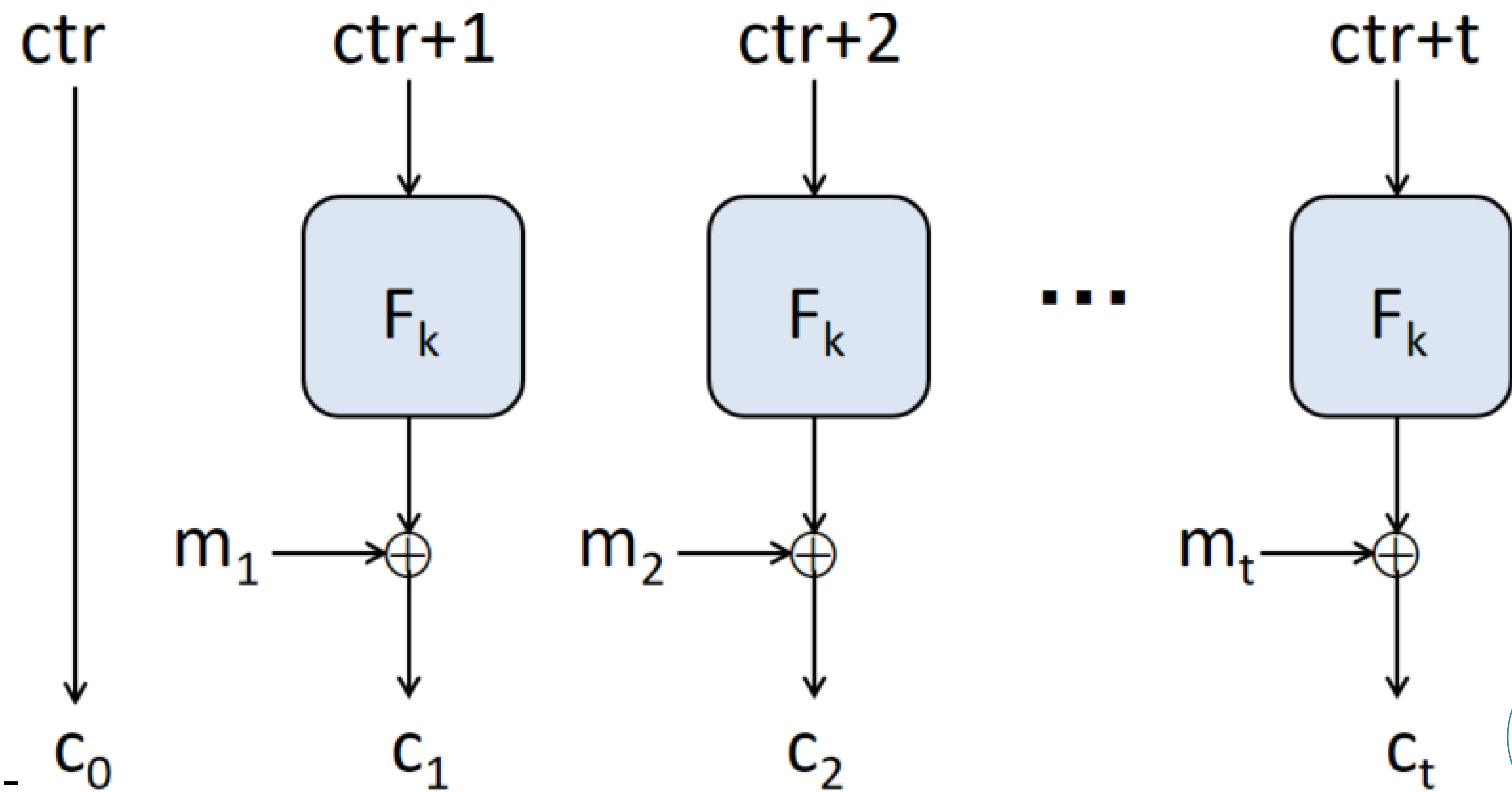
- The *birthday bound* comes up in many other cryptographic contexts

- **Example**: *IV* reuse in CTR-mode encryption
  - If k messages are encrypted, what is the chance that some *IV* is used twice?
  - Note: this is much higher than the probability that a *specific* *IV* is used again

"Birthday bound"

- The *birthday bound* comes up in many other cryptographic contexts

- **Example**: *IV* reuse in CTR-mode encryption



Collision resistent hash functions

- CRH vs. PRG
  - CRHs are dual to PRGs, in the sense that the goal is to *shrink* the input as much as possible. Similar to PRGs, if one can shrink the input by even by one bit, one can get a CRH collection that shrinks the bits by an amount of polynoimial.
  - In practice, people usually talk about a *single* hash func. rather than a collection. One can think of this that someone chose the key k once and then fixed the function h_k for everyone to use. In fact, most practical constructions involve some hardwired standardized constants, often known as *IV* that can be thought of as a choice of the key.

Construction of CRH

- Practical constructions of cryptographic hash functions start with a basic block, a.k.a. compression function $h : \{0,1\}^{2n} \to \{0,1\}^n$. The function $H : \{0,1\}^* \to \{0,1\}^n$ is defined as
$$H(m_1, \ldots, m_t) = h(h(h(m_1, IV), m_2), \ldots, m_t),$$
when the message is composed of t blocks (and we can pad it otherwise). This is known as the *Merkle-Damgård construction*.

Construction of CRH

- Practical constructions of cryptographic hash functions start with a basic block, a.k.a. compression function $h : \{0,1\}^{2n} \to \{0,1\}^n$. The function $H : \{0,1\}^* \to \{0,1\}^n$ is defined as
  $$H(m_1, \ldots, m_t) = h(h(h(m_1, IV), m_2), \ldots, m_t),$$
  when the message is composed of t blocks (and we can pad it otherwise). This is known as the *Merkle-Damgård construction*.

- **Theorem 8.1** (Merkle-Damgård preserves collision resistance)
  Let H be constructed from h above. Then given two messages $m \neq m' \in \{0,1\}^{tn}$ such that $H(m) = H(m')$ we can efficiently find two messages $x \neq x' \in \{0,1\}^{2n}$ such that $h(x) = h(x')$.

Construction of CRH

- **Theorem 8.1** (Merkle-Damgård preserves collision resistance)
  Let H be constructed from h above. Then given two messages $m \neq m' \in \{0,1\}^{tn}$ such that $H(m) = H(m')$ we can efficiently find two messages $x \neq x' \in \{0,1\}^{2n}$ such that $h(x) = h(x')$.

  Proof. The intuition behind the proof is that if h is invertible then we could invert H by simply going backwards.

$$H(m_1, \ldots, m_t) = h(h(h(m_1, IV), m_2), \ldots, m_t),$$

Construction of CRH

- **Theorem 8.1** (Merkle-Damgård preserves collision resistance)
  Let H be constructed from h above. Then given two messages $m \neq m' \in \{0,1\}^{tn}$ such that $H(m) = H(m')$ we can efficiently find two messages $x \neq x' \in \{0,1\}^{2n}$ such that $h(x) = h(x')$.

  Proof. The intuition behind the proof is that if h is invertible then we could invert H by simply going backwards.

  In principle, if a collision for H exists then so does a collision for h.

$$H(m_1, \ldots, m_t) = h(h(h(m_1, IV), m_2), \ldots, m_t),$$

Construction of CRH

- **Theorem 8.1** (Merkle-Damgård preserves collision resistance)
  Let H be constructed from h above. Then given two messages $m \neq m' \in \{0,1\}^{tn}$ such that $H(m) = H(m')$ we can efficiently find two messages $x \neq x' \in \{0,1\}^{2n}$ such that $h(x) = h(x')$.

  Proof. The intuition behind the proof is that if h is invertible then we could invert H by simply going backwards.

  In principle, if a collision for H exists then so does a collision for h.

  We look at the computation of $H(m)$ and $H(m')$ and at the first block in which the inputs differ but the output is the same (there must be such a block). This block will yield a collision for h.

$$H(m_1, \ldots, m_t) = h(h(h(m_1, IV), m_2), \ldots, m_t),$$

Hash functions in practice

- MD5
  - Developed in 1991 by Ron Rivest
  - 128-bit output length
  - Collisions found in 2004, and more recently in 2013, should no longer be used

Hash functions in practice

- MD5
  - Developed in 1991 by Ron Rivest
  - 128-bit output length
  - Collisions found in 2004, and more recently in 2013, should no longer be used
- SHA-1
  - Introduced in 1995 by NSA
  - 160-bit output length
  - Theoretical analysis indicates some weaknesses
  - Current trend to migrate to SHA-2
  - Collision found by brute force in 2017!

Hash functions in practice

- SHA-2
  - Supports 224, 256, 384, and 512-bit outputs
  - No serious known weaknesses

Hash functions in practice

- SHA-2
  - Supports 224, 256, 384, and 512-bit outputs
  - No serious known weaknesses
- SHA-3 / Keccak
  - Result of a public competition from 2008-2012
  - Very different design from SHA-1/SHA-2
  - Supports 224, 256, 384, and 512-bit outputs

Recall

- We showed how to construct a *secure MAC* for short, fixed-length messages based on any PRF/block cipher

Recall

- We showed how to construct a *secure MAC* for short, fixed-length messages based on any PRF/block cipher

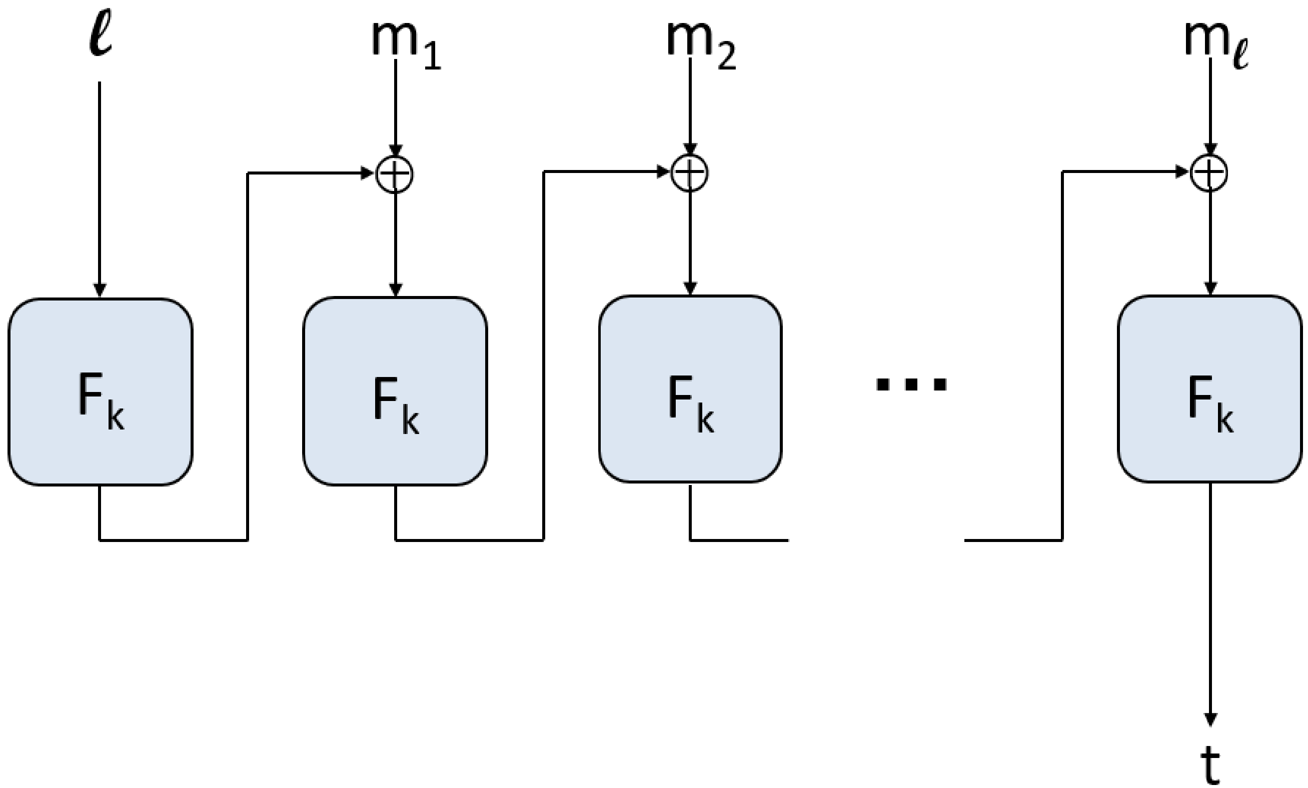
  Construct the following MAC Π:
  - Gen: choose a uniform key k for F
  - $Mac_k(m)$: output $F_k(m)$
  - $Vrfy_k(m, t)$: output 1 iff $F_k(m) = t$

Recall

- We showed how to construct a *secure MAC* for short, fixed-length messages based on any PRF/block cipher
- We want to extend this to a *secure MAC* for arbitrary-length messages
  - Before: using CBC-MAC

Recall

- We showed how to construct a *secure MAC* for short, fixed-length messages based on any PRF/block cipher
- We want to extend this to a *secure MAC* for arbitrary-length messages
  - Before: using CBC-MAC



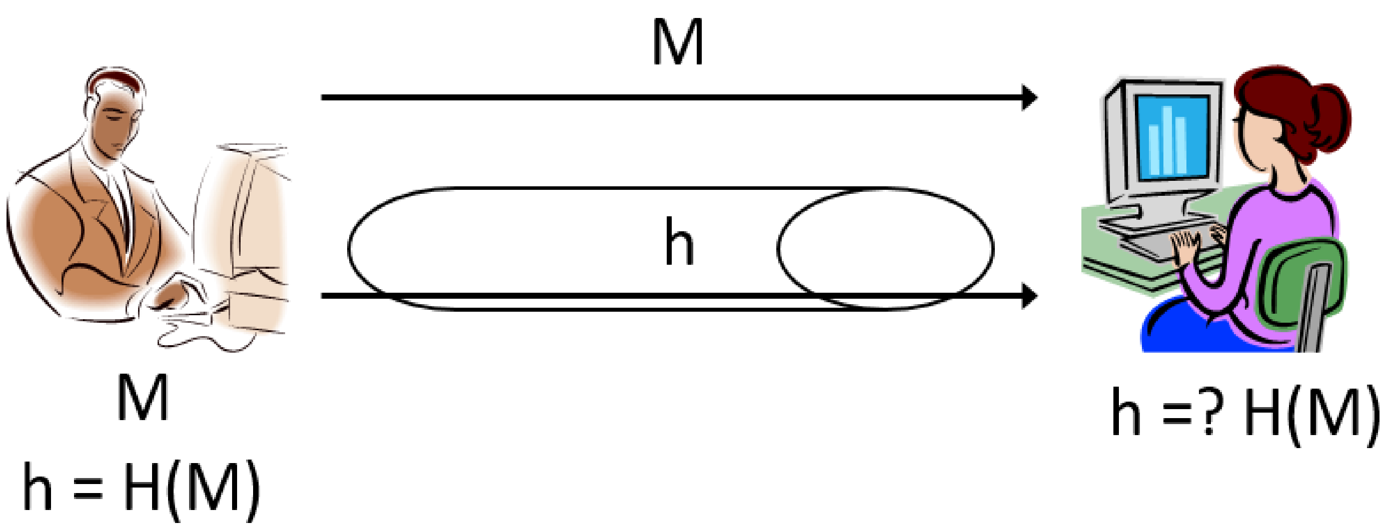
Recall

- We showed how to construct a *secure MAC* for short, fixed-length messages based on any PRF/block cipher

- We want to extend this to a *secure MAC* for arbitrary-length messages
  - Before: using CBC-MAC
  - After: using hash functions

Recall

- We showed how to construct a *secure MAC* for short, fixed-length messages based on any PRF/block cipher
- We want to extend this to a *secure MAC* for arbitrary-length messages
  - Before: using CBC-MAC
  - After: using hash functions



Hash-and-MAC

k

M

$h = H(M)$

$t = \mathrm{Mac}_k(h)$

$\xrightarrow{M}$

$\xrightarrow{h,\ t}$

k

$h = H(M)$

$\mathrm{Vrfy}_k(h, t) = 1?$

Security

- **Theorem 8.2** If the *MAC* is *secure* for fixed-length messages and H is *collision-resistent*, then the previous construction is a *secure* MAC for arbitrary-length messages

Security

- **Theorem 8.2** If the *MAC* is *secure* for fixed-length messages and H is *collision-resistent*, then the previous construction is a *secure* MAC for arbitrary-length messages
- Proof sketch
  Say the sender authenticates $M_1, M_2, \ldots$
  - Let $m_i = H(M_i)$

Security

- **Theorem 8.2** If the *MAC* is *secure* for fixed-length messages and H is *collision-resistent*, then the previous construction is a *secure* MAC for arbitrary-length messages
- Proof sketch
  Say the sender authenticates $M_1, M_2, \ldots$
  - Let $m_i = H(M_i)$

  Attacker outputs forgery (M, t), $M \neq M_i$ for all i

Security

- **Theorem 8.2** If the *MAC* is *secure* for fixed-length messages and H is *collision-resistent*, then the previous construction is a *secure* MAC for arbitrary-length messages
- Proof sketch
  Say the sender authenticates M_1, M_2, ...
  - Let $m_i = H(M_i)$

  Attacker outputs forgery (M, t), $M \neq M_i$ for all i

  Two cases:
  - $H(M) = H(M_i)$ for some i
    - Collision in H!

Security

- **Theorem 8.2** If the *MAC* is *secure* for fixed-length messages and H is *collision-resistent*, then the previous construction is a *secure* MAC for arbitrary-length messages
- Proof sketch
  Say the sender authenticates $M_1, M_2, \ldots$
  - Let $m_i = H(M_i)$

  Attacker outputs forgery (M, t), $M \neq M_i$ for all i

  Two cases:
  - $H(M) = H(M_i)$ for some i
    - Collision in H!
  - $H(M) \neq m_i$ for all i
    - Forgery in the underlying, fixed-length MAC!

Instantiation

- Hash function + block-cipher-based MAC
  - Block-length mismatch
  - Need to implement two crypto primitives (block cipher and hash function)

Instantiation

- Hash function + block-cipher-based MAC
  - Block-length mismatch
  - Need to implement two crypto primitives (block cipher and hash function)
- HMAC: constructed entirely from (certain type of) hash functions
  - MD5, SHA-1, SHA-2
  - Not SHA-3

Instantiation

- Hash function + block-cipher-based MAC
  - Block-length mismatch
  - Need to implement two crypto primitives (block cipher and hash function)
- HMAC: constructed entirely from (certain type of) hash functions
  - MD5, SHA-1, SHA-2
  - Not SHA-3
- Can be viewed as following the hash-and-MAC paradigm
  - With (part of the) hash function being used as a PRF

Other applications of hash functions

- Hash functions are ubiquitous
  - Collision-resistance $\Rightarrow$ "fingerprinting"
  - Used as a *one-way function*
  - Used as a "random oracle"
    - Proofs of work

Other applications of hash functions

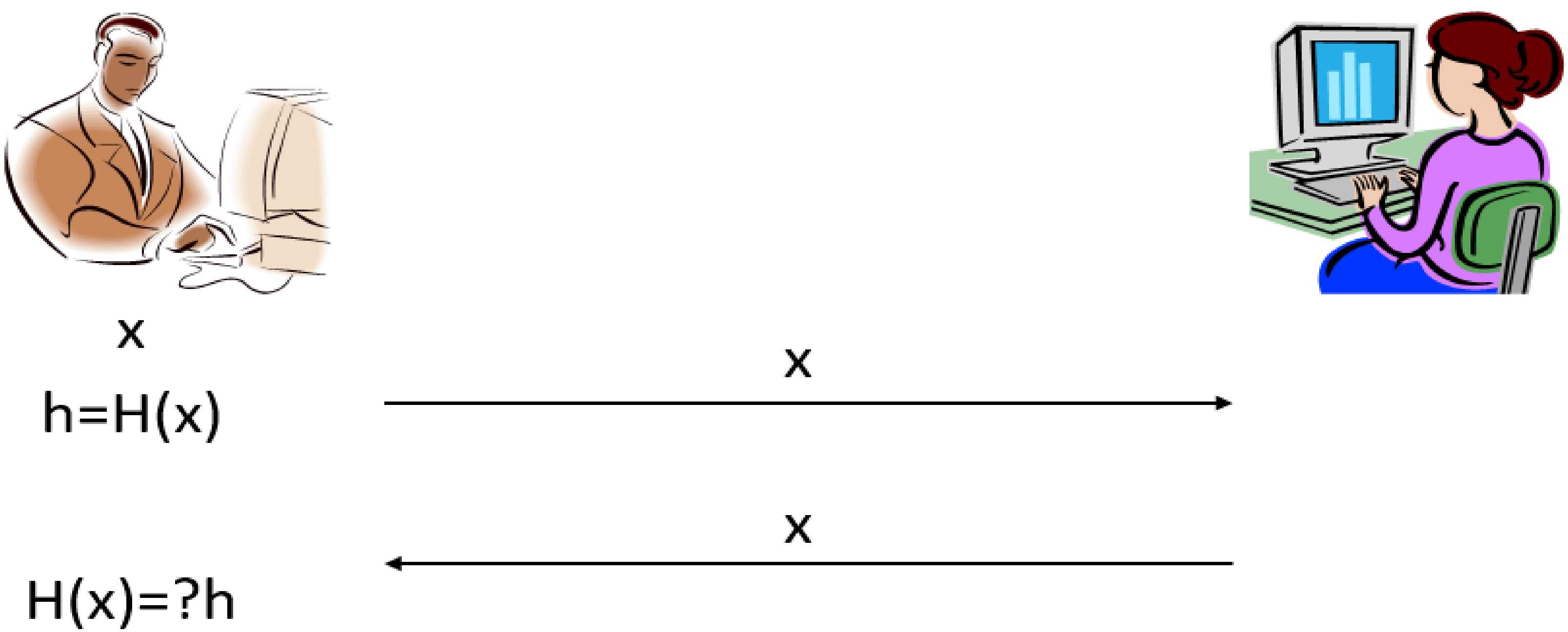
- Hash functions are ubiquitous
  - Collision-resistance $\Rightarrow$ "fingerprinting"
  - Used as a *one-way function*
  - Used as a "random oracle"
    - Proofs of work
  - E.g., virus scanning
  - E.g., deduplication

Fingerprinting

- E.g., file integrity
  - Assuming it is possible to get a reliable copy of $H(x)$ for file x
  - Note: different from integrity in the context of message-authentication codes

Fingerprinting

- E.g., file integrity
  - Assuming it is possible to get a reliable copy of $H(x)$ for file x
  - Note: different from integrity in the context of message-authentication codes
- How to outsource files to an untrusted server?



Outsourced storage

$x_1, \ldots, x_n$

$h_i = H(x_i)$

$H(x_i) =? h_i$

$x_1, \ldots, x_n$

i

x_i

O(n) client storage!

Outsourced storage

$x_1, \ldots, x_n$

$h = H(x_1, \ldots, x_n)$

$\xrightarrow{x_1, \ldots, x_n}$

$\xrightarrow{i}$

$\xleftarrow{x_1, \ldots, x_n}$

$H(x_1, \ldots, x_n) =? h$

$O(n \cdot |x|)$ communication!

Outsourced storage

$x_1, \ldots, x_n$

$h = H(H(x_1), \ldots, H(x_n))$

$\xrightarrow{x_1, \ldots, x_n}$

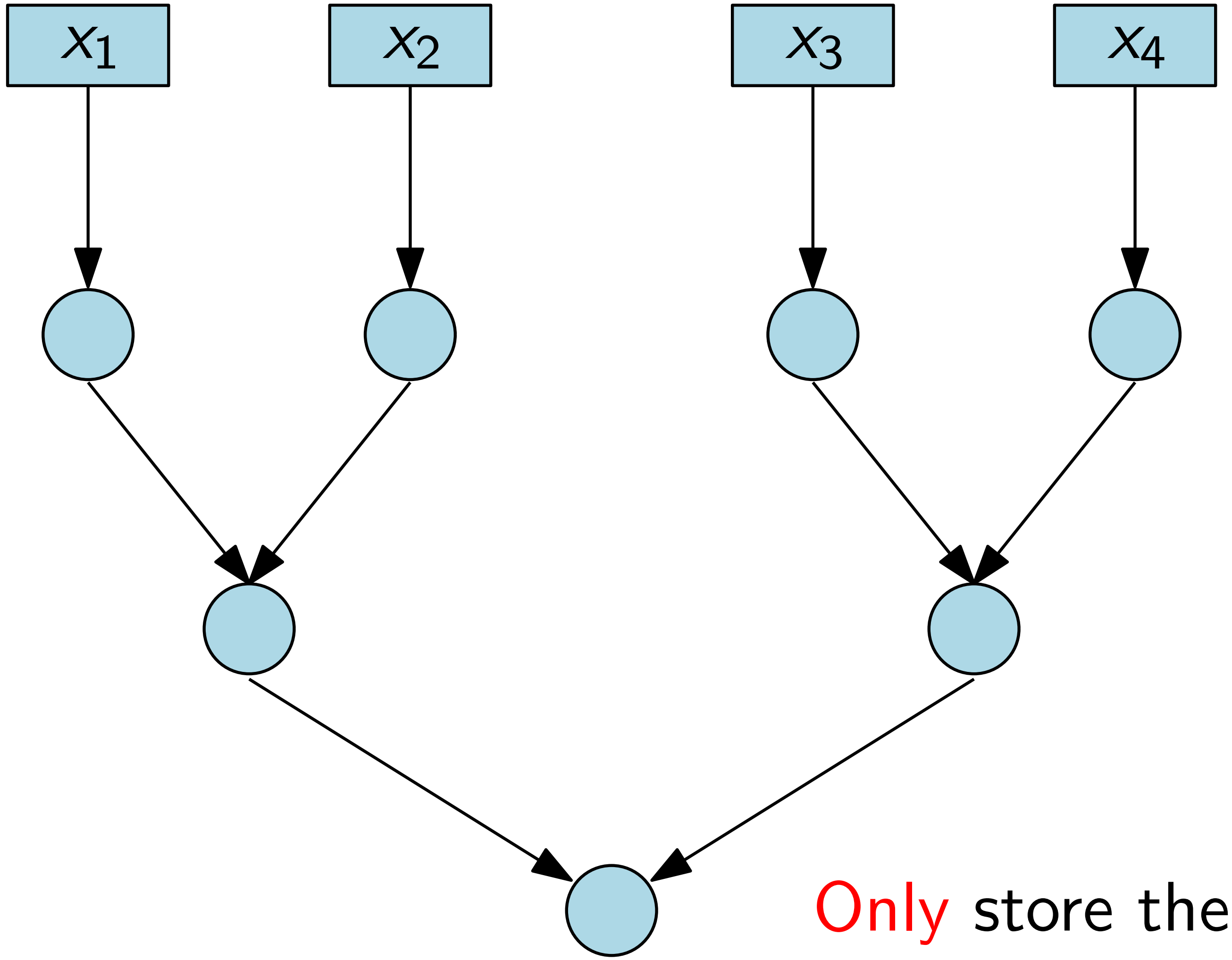
$\xrightarrow{i}$

$\xleftarrow{x_i, h_1, \ldots, h_n}$

$H(h_1, \ldots, H(x_i), \ldots, h_n) =? h$

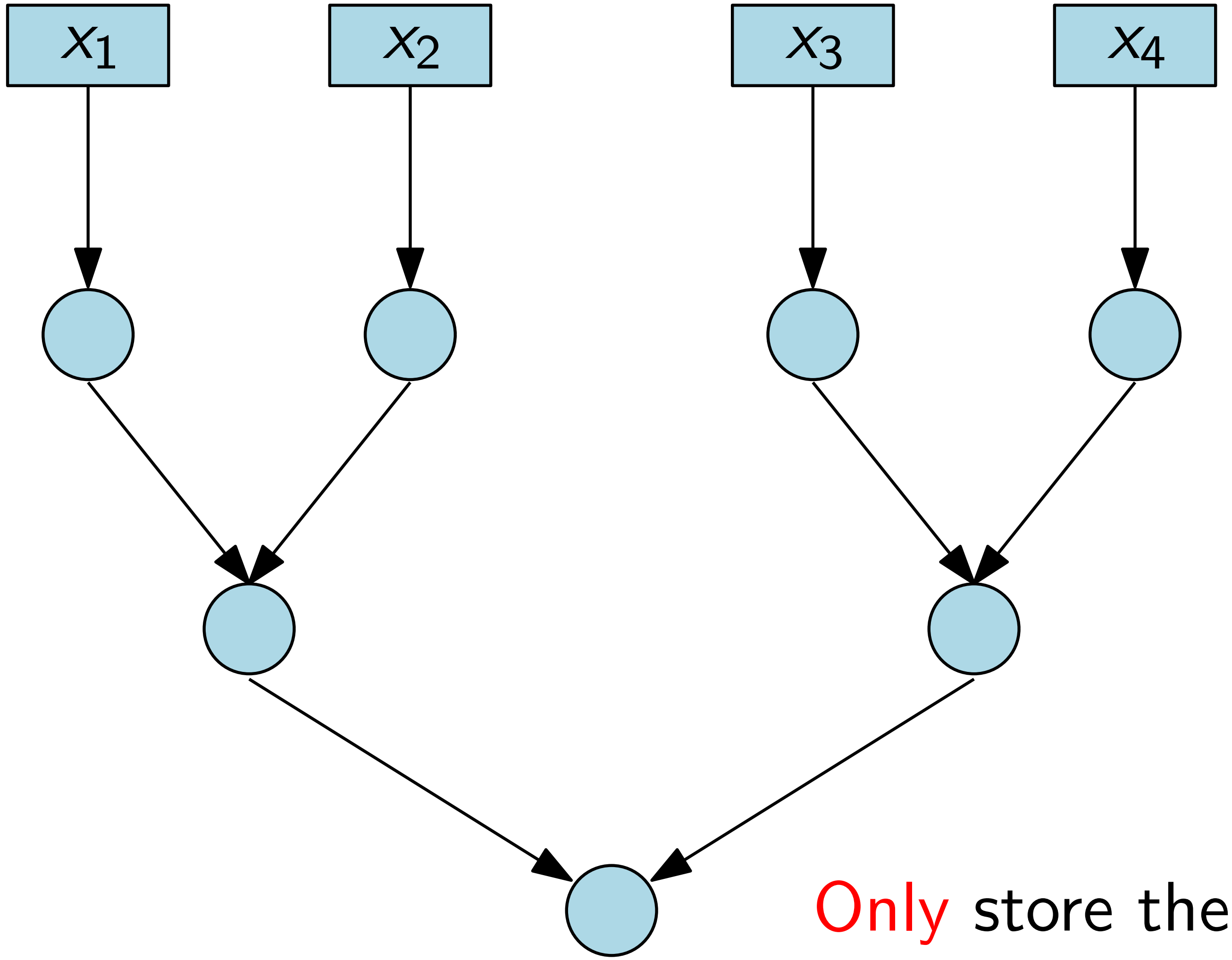
$|x_i|$ + O(n) communication!

Merkle tree



Only store the root!

Merkle tree



Only store the root!

$O(\log n)$ communication/computation!

Outsourced storage

$x_1, \ldots, x_n$

$h = H(H(x_1), \ldots, H(x_n))$ $\xrightarrow{x_1, \ldots, x_n}$

$\xrightarrow{i}$

$\xleftarrow{x_i, h_1, \ldots, h_n}$

$H(h_1, \ldots, H(x_i), \ldots, h_n) =? h$

$|x_i| + O(n)$ communication!

- Using a Merkle tree, we can solve the outsourcing problem with $O(1)$ client storage and $|x| + O(\log n)$ communication.

Key derivation

- Consider deriving a (shared) key from (shared) high-entropy information
  - E.g., biometric data
  - E.g., generating randomness

Key derivation

- Consider deriving a (shared) key from (shared) high-entropy information
  - E.g., biometric data
  - E.g., generating randomness

- Cryptographic keys must be *uniform*, but shared data is only high-entropy

Min-entropy

- Let X be a distribution
- The *min-entropy* of X (measured in bits) is

  $$H_\infty(X) = -\log \max_x\{\Pr[X = x]\}$$

  – I.e., if $H_\infty(X) = n$, then the probability of guessing x sampled from X is (at most) 2^{-n}
- Min-entropy is more suitable for crypto than entropy

Min-entropy

- Let X be a distribution
- The *min-entropy* of X (measured in bits) is

  $$H_\infty(X) = -\log \max_x\{\Pr[X = x]\}$$

  – I.e., if $H_\infty(X) = n$, then the probability of guessing x sampled from X is (at most) 2^{-n}
- Min-entropy is more suitable for crypto than entropy
- Given shared information x (sampled from distribution X), derive shared key $k = H(x)$
  – In what sense can we claim that k is a "good" (i.e., uniformly distributed) cryptographic key?

Private-key schemes

- We have seen how to construct schemes based on various lower-level primitives
  - Stream ciphers / PRGs
  - Block ciphers / PRFs
  - Hash functions

Private-key schemes

- We have seen how to construct schemes based on various lower-level primitives
  - Stream ciphers / PRGs
  - Block ciphers / PRFs
  - Hash functions

- How do we construct these primitives?

Two approaches

- Construct from even lower-level assumptions
  - Can prove secure (given lower-level assumption)
  - Typically inefficient

Two approaches

- Construct from even lower-level assumptions
  - Can prove secure (given lower-level assumption)
  - Typically inefficient

- Build directly
  - Much more efficient!
  - Need to assume security, but
    - We have formal definitions to aim for
    - We can concentrate our analysis on these primitives
    - We can develop/analyze various design principles

Terminology

- *Init* algorithm
  - Takes as input a key + initialization vector (*IV*)
  - Outputs initial state
- *GetBits* algorithm
  - Takes as input the current state
  - Outputs next bit/byte/chunk and updated state
  - Allows generation of as many bits as needed

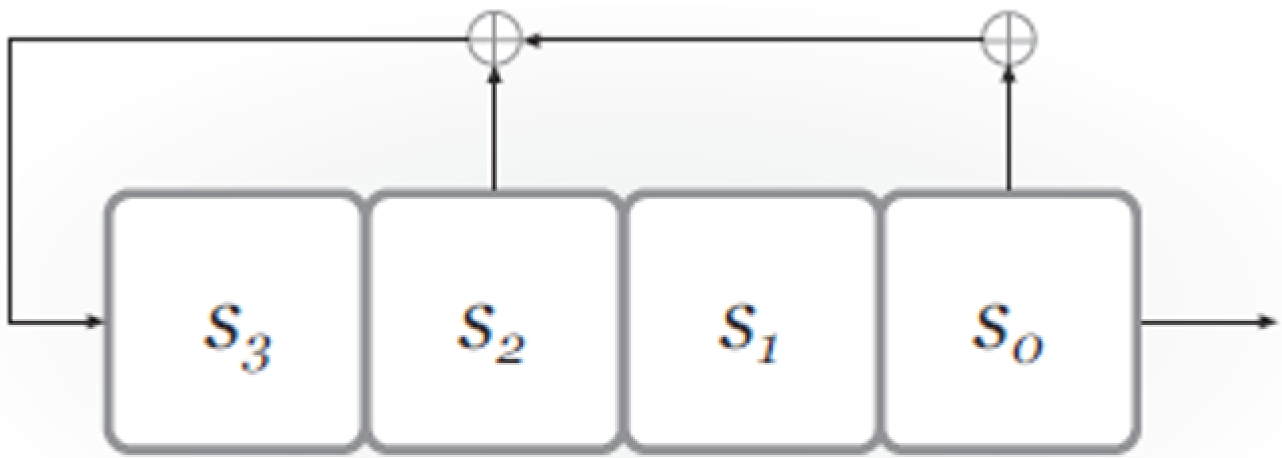
Security requirements

- If there is no *IV*, then (for a uniform key) the output of *GetBits* should be indistinguishable from a uniform, independent stream of bits

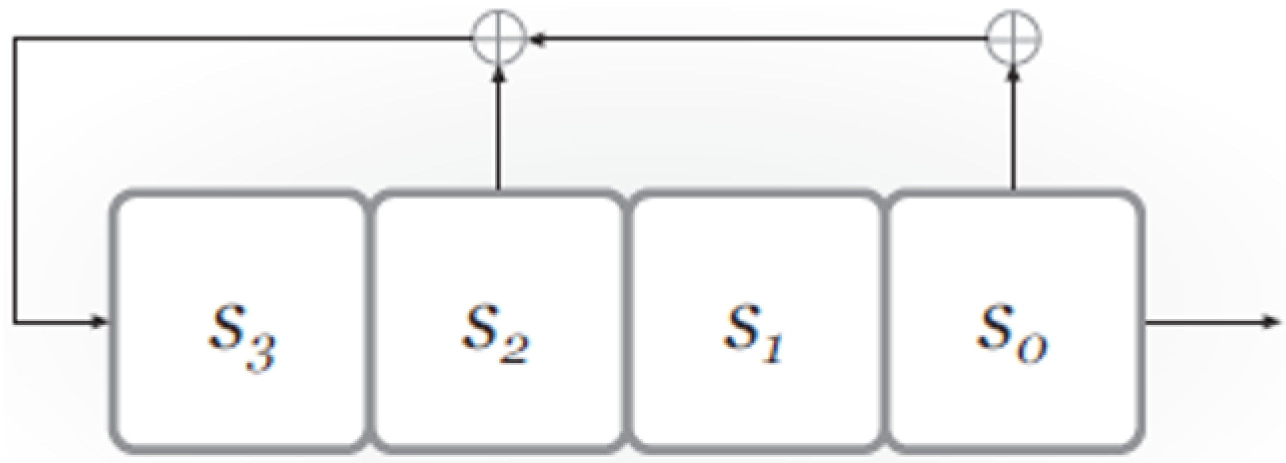
- If there is an *IV*, then (for a uniform key) the output of *GetBits* on multiple, uniform *IV*s should be indistinguishable from multiple uniform, independent streams of bits
  - Even if the attacker is given the *IV*s

LFSRs

- Degree $n \Rightarrow n$ registers
- State: bits $s_{n-1}, \ldots, s_0$ (contents of the registers)
- Feedback coefficients $c_{n-1}, \ldots, c_0$ (view as part of state; do not change)
- State updated and output generated in each "clock tick"



Example



- Assume initial content of registers is 0100
- First 4 state transitions:
  0100 $\to$ 1010 $\to$ 0101 $\to$ 0010 $\to$ ...
- First 3 output bits:
  0 0 1 ...

LFSRs as stream ciphers

- Key + IV used to initialize the state of the LFSR (possibly including feedback coefficients)
- One bit of output per clock tick
  - State updated

LFSRs as stream ciphers

- Key + *IV* used to initialize the state of the LFSR (possibly including feedback coefficients)
- One bit of output per clock tick
  - State updated
- State (and output) "cycles" if state ever repeated
- *Maximal-length LFSR* cycles through all $2^n - 1$ nonzero states
  - Known how to set feedback coefficients so as to achieve maximal length

LFSRs as stream ciphers

- Key + *IV* used to initialize the state of the LFSR (possibly including feedback coefficients)
- One bit of output per clock tick
  - State updated
- State (and output) "cycles" if state ever repeated
- *Maximal-length LFSR* cycles through all $2^n - 1$ nonzero states
  - Known how to set feedback coefficients so as to achieve maximal length
- *Maximal-length LFSR*s have good statistical properties, but they are not cryptographically secure!

Security of LFSRs

- If feedback coefficients known ,the first n output bits directly reveal the initial state!

Security of LFSRs

- If feedback coefficients known ,the first n output bits directly reveal the initial state!
- Even if feedback coefficients are unknown, can use linear algebra to learn everything from $2n$ consecutive output bits (*Berlekamp-Massey algorithm*)

Security of LFSRs

- If feedback coefficients known ,the first n output bits directly reveal the initial state!
- Even if feedback coefficients are unknown, can use linear algebra to learn everything from $2n$ consecutive output bits (*Berlekamp-Massey algorithm*)
- Linearity is *bad* for cryptography (because linear algebra is so powerful)

Nonlinear FSRs

- Add *nonlinearity* to prevent attacks
  - Nonlinear feedback
  - Output is a nonlinear function of the state
  - Multiple (coupled) LFSRs
  - or any combination of the above

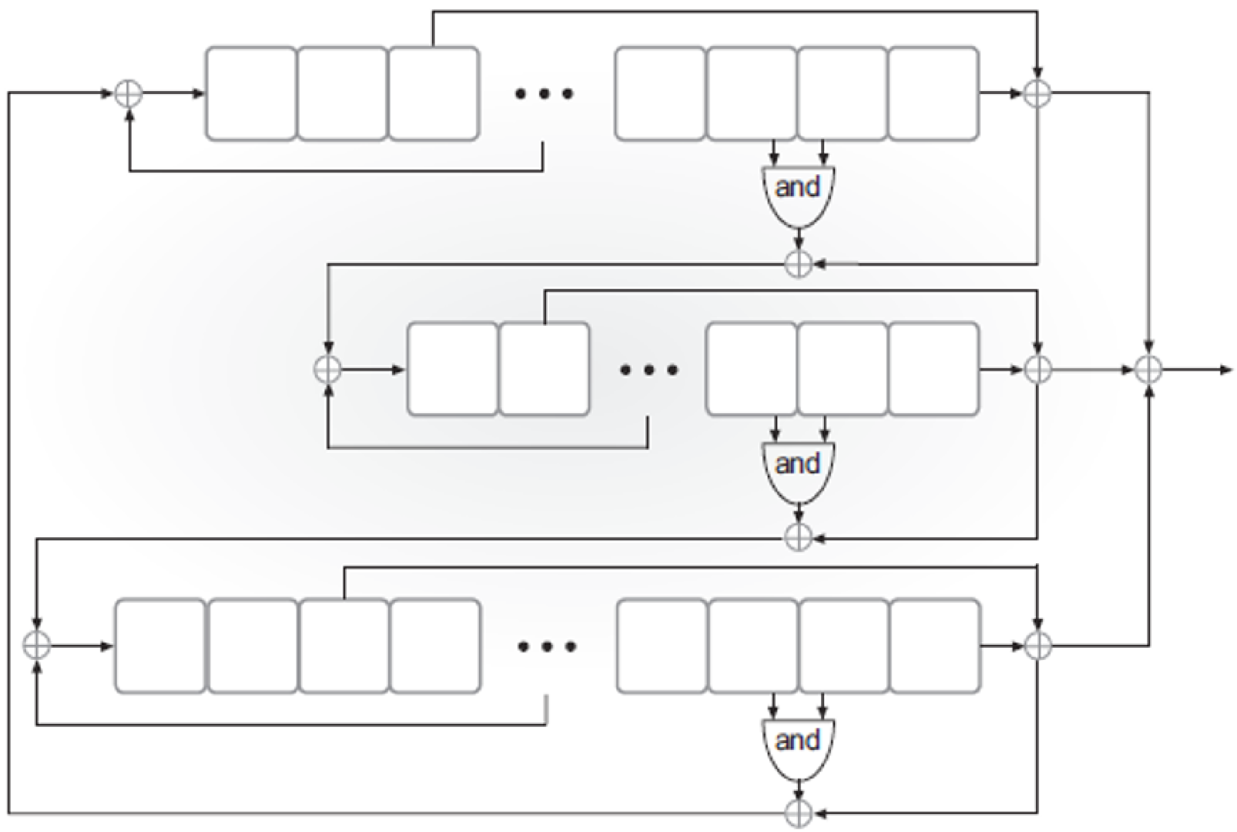
Nonlinear FSRs

- Add *nonlinearity* to prevent attacks
  - Nonlinear feedback
  - Output is a nonlinear function of the state
  - Multiple (coupled) LFSRs
  - or any combination of the above

- Still want to preserve statistical properties of the output, and long cycle length

Example: Trivium

- Designed by De Canniere and Preneel in 2006 as part of eSTREAM competition
- Intended to be simple and efficient (especially in hardware)
- Essentially no attacks better than brute-force search are known



Example: Trivium

- Three FSRs of degree 93, 84, and 111

Example: Trivium

- Three FSRs of degree 93, 84, and 111
- Initialization:
  - 80-bit key in left-most registers of first FSR
  - 80-bit *IV* in left-most registers of second FSR
  - Remaining registers set to 0, except for three right-most registers of third FSR
  - Run for 4×288 clock ticks

Next Lecture

- more on private-key schemes ...