PennState

CSE 443: Introduction to Computer Security

Module: Hashing

Prof. Syed Rafiul Hussain
Department of Computer Science and Engineering
The Pennsylvania State University

Acknowledgements: Some of the slides have been adopted from
Trent Jaeger (Penn State), Patrick McDaniel (Penn State), Ninghui Li (Purdue) and William Enck (NCSU)

Data Integrity and Source Authentication

- Encryption does not protect data from modification by another party.
- Need a way to ensure that data arrives at destination in its original form as sent by the sender and it is coming from an authenticated source.

Hash Functions

- •A hash function maps a message of an arbitrary length to a m-bit output output known as the fingerprint or the message digest
  - if the message digest is transmitted securely, then changes to the message can be detected

- A hash function is a many-to-one function, so collisions can happen.

Hash Algorithms

- Hash algorithm
  - Compression of data into a hash value
  - E.g., h(d) = parity(d)
  - Such algorithms are generally useful in algorithms (speed/space optimization)
- … as used in cryptosystems
  - *Given a function $h:X \rightarrow Y$, then we say that h is:*
    - *preimage resistant (one-way):*
      - *if given $y \in Y$ it is computationally infeasible to find a value $x \in X$ s.t. $h(x) = y$*
    - *2-nd preimage resistant (weak collision resistant):*
      - *if given $x \in X$ it is computationally infeasible to find a value $x' \in X$, s.t. $x'^{1}x$ and $h(x') = h(x)$*
    - *collision resistant (strong collision resistant):*
  - *if it is computationally infeasible to find two distinct values $x', x \in X$, s.t. $h(x') = h(x)$*

Consequences of These Properties

- •For a good cryptographic hash, if you change one bit of the input, the output should change drastically and unpredictably
  - MD5("ABCD") = 0xed5d34c74e59d16bd6d5b3683db655c3
  - MD5("ABCE") = 0x95741cb5c4ee614792f6f5a44f2e107a
- So if you need to know if a file has changed, hash it!
-

Uses of hash functions

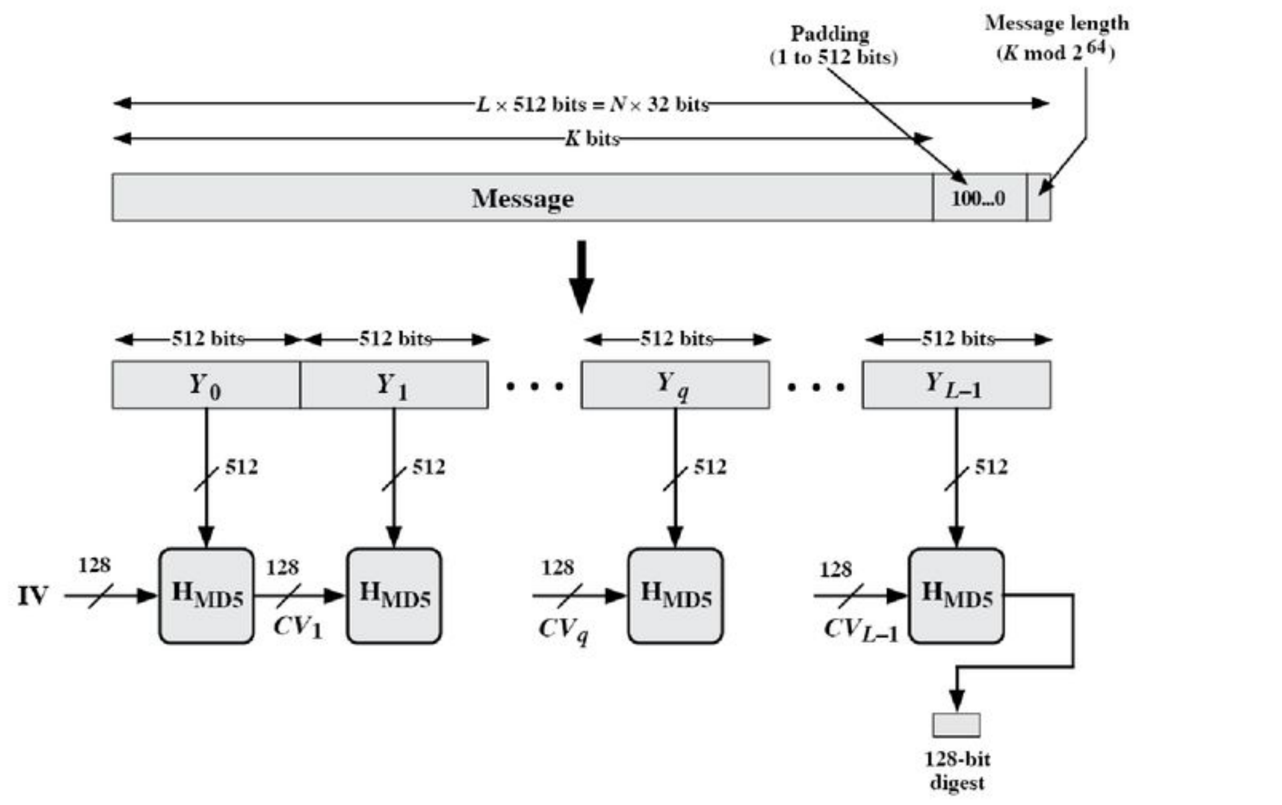
- Software integrity
  - E.g., tripwire
- Timestamping
  - How?

- Message authentication
- One-time Passwords
- Digital signature

Hash Functions

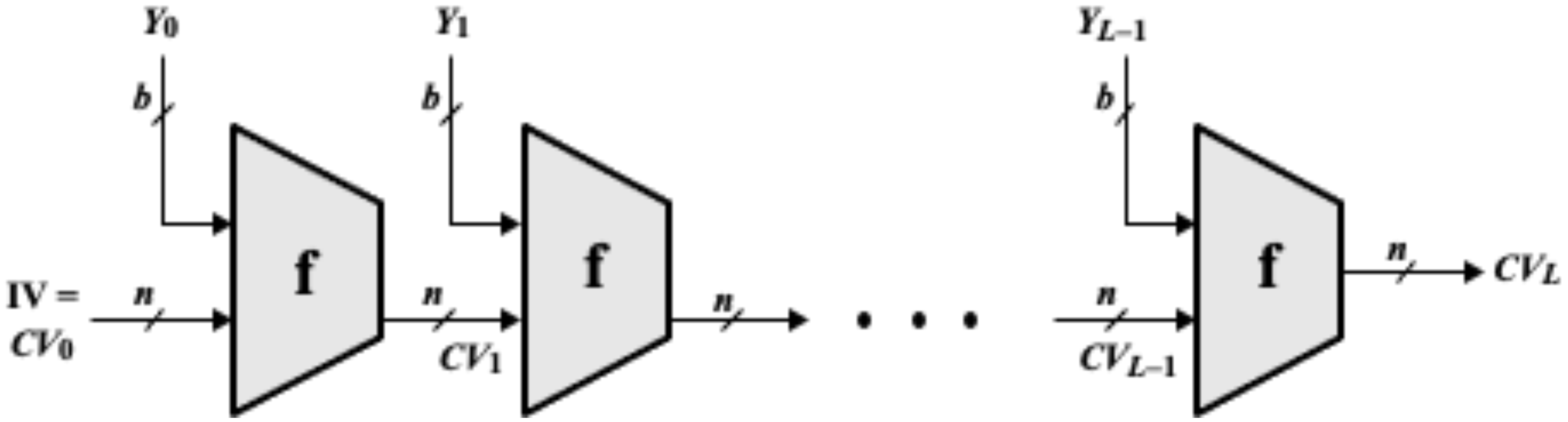
- MD4, MD5
  - Substitution on complex functions in multiple passes
- SHA-1
  - 160-bit hash
  - “Complicated function”
- SHA-2, 2001
  - 256 to 512 bit hash (SHA-256)
- SHA-3, 2015
  - Keccak Algorithm
- Limited formal basis
  - Practical attacks on SHA-1, MD5

MD5



Message Digest Generation Using MD5 15

General Structure of Hash



IV	=	Initial value	L	=	number of input blocks
CV_i	=	chaining variable	n	=	length of hash code
Y_i	=	ith input block	b	=	length of input block
f	=	compression algorithm			

Using hashes as authenticators

- Consider the following scenario
  - Prof. Alice has not decided if she will cancel the next lecture.
  - When she does decide, she communicates to Bob the student through Mallory, her evil TA.
  - She does not care if Bob shows up to a cancelled class
  - She wants Bob to show for all classes held
- She and Bob use the following protocol:

1. Alice invents a secret t
2. Alice gives Bob h(t), where h() is a crypto hash function
3. If she cancels class, she gives t to Mallory to give to Bob

– If does not cancel class, she does nothing

– If Bob receives the token t, he knows that Alice sent it

Hash Authenticators

- Why is this protocol secure?
  - t acts as an authenticated value (authenticator) because Mallory could not have produced t without inverting h()
  - Note: Mallory can convince Bob that class is occurring when it is not by simply not delivering t (but we assume Bob is smart enough to come to that conclusion when the room is empty)

- What is important here is that hash preimages are good as (single bit) authenticators.
- Note that it is important that Bob got the original value h(t) from Alice directly (was provably authentic)

Hash chain

- Now, consider the case where Alice wants to do the same protocol, only for all 26 classes (the semester)
- Alice and Bob use the following protocol:

1. Alice invents a secret t
2. Alice gives Bob $h^{26}(t)$, where $h^{26}()$ is 26 repeated uses of $h()$.
3. If she cancels class on day d, she gives $h^{(26-d)}(t)$ to Mallory, e.g.,

   If cancels on day 1, she gives Mallory $h^{25}(t)$

   If cancels on day 2, she gives Mallory $h^{24}(t)$

   …….

   If cancels on day 25, she gives Mallory $h^{1}(t)$

   If cancels on day 26, she gives Mallory t
4. If does not cancel class, she does nothing

– If Bob receives the token t, he knows that Alice sent it

Hash Chain (cont.)

- Why is this protocol secure?
  - On day d, $h^{(26-d)}(t)$ acts as an authenticated value (authenticator) because Mallory could not create $h^{(26-d)}(t)$ without inverting $h^{(26-d-1)}(t)$ because for any $h^k(t)$ she has $h^j(t)$ where $26>j>k$
  - That is, Mallory potentially has access to the hash values for all days prior to today, but that provides no information on today's value, as they are all post-images of today's value
  - Note: Mallory can again convince Bob that class is occurring by not delivering $h^{(26-d)}(t)$
  - Chain of hash values are ordered authenticators
- Important that Bob got the original value $h^{26}(t)$ from Alice directly (was provably authentic)

A (simplified) sample token device

- A one-time password system that essentially uses a *hash chain* as authenticators.
  - For seed (S) and chain length (l), epoch length (x)
  - Tamperproof token encodes S in firmware

$$\mathrm{pw}_i = h^{l-i}(S)$$

  - Device display shows password for epoch i
  - Time synchronization allows authentication server to know what i is expected, and authenticate the user.
- *Note*: somebody can see your token display at some time but learn nothing useful for later periods.

Birthday Paradox

- Q: Why is the birthday paradox important to hash functions?

- **Birthday paradox :** the probability that two or more people in a group of 23 share the same birthday is >than 50%

Compute $P(A)$: probability that at least two people in the room have the same birthday.

Compute $P(A')$: the probability that no two people in the room have the same birthday.

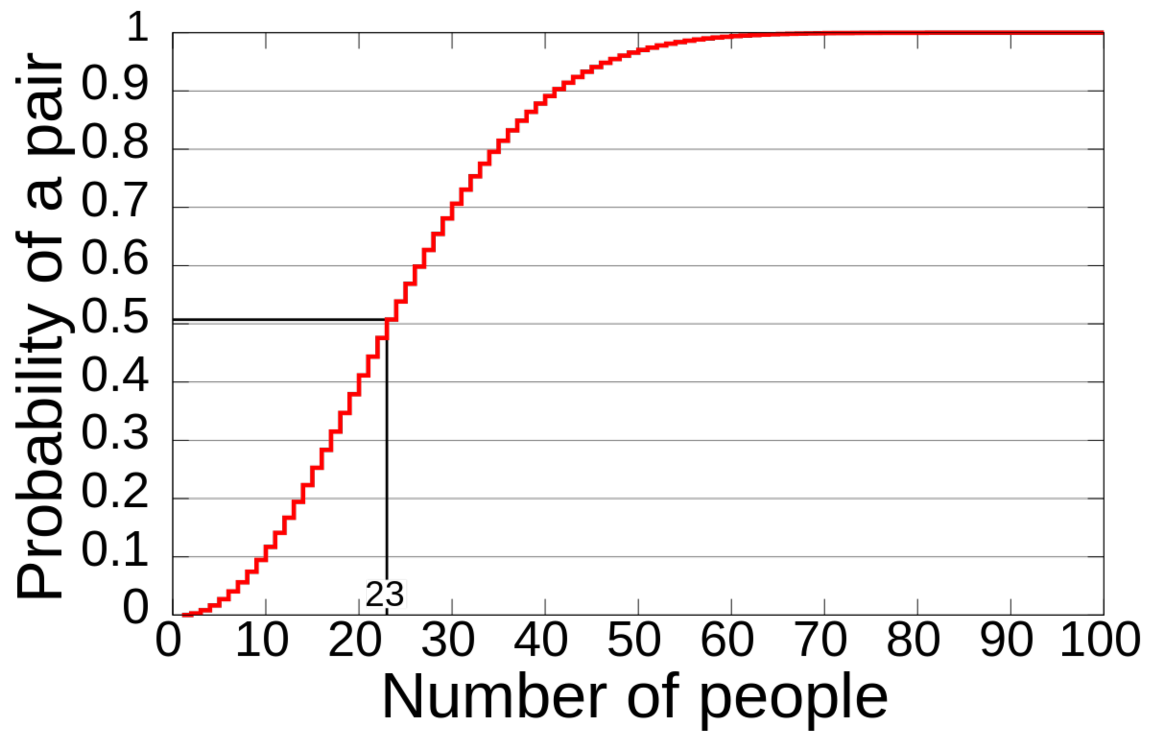
$$P(A') = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \frac{362}{365} \times \cdots \times \frac{343}{365}$$

The terms of equation (1) can be collected to arrive at:

$$P(A') = \left(\frac{1}{365}\right)^{23} \times (365 \times 364 \times 363 \times \cdots \times 343)$$

Evaluating equation (2) gives $P(A') \approx 0.492703$

Therefore, $P(A) \approx 1 - 0.492703 = 0.507297$ (50.7297%).



$$\bar{p}(n) = 1 \times \left(1 - \frac{1}{365}\right) \times \left(1 - \frac{2}{365}\right) \times \cdots \times \left(1 - \frac{n-1}{365}\right)$$

$$= \frac{365 \times 364 \times \cdots \times (365 - n + 1)}{365^n}$$

$$= \frac{365!}{365^n (365-n)!} = \frac{n! \cdot \binom{365}{n}}{365^n} = \frac{{}_{365}P_n}{365^n}$$

where ! is the factorial operator, $\binom{365}{n}$ is the binomial coefficient and ${}_kP_r$ denotes permutation.

Message Authentication Code

- MAC
  - Used in protocols to authenticate content, authenticates integrity for data d
  - To simplify, hash function h(), key k, data d

$$MAC(k,m) = h(k \,||\, m)$$

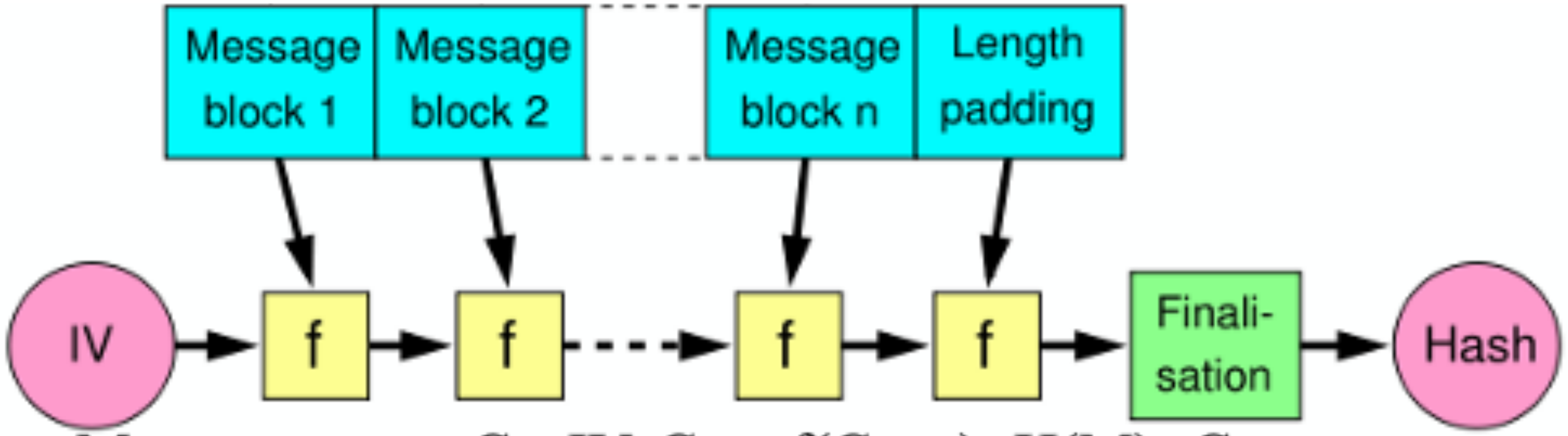
  - E.g., XOR the key with the data and hash the result

- Q: Why does this provide integrity?
  - Cannot produce MAC(k,m) unless you know k
  - If you could, then can *invert* h()
- Exercise for class: prove the previous statement

Constructing MAC from Hash

- Message is divided into fixed-size blocks and padded
- Uses a compression function f, which takes a chaining variable (of size of hash output) and a message block, and outputs the next chaining variable
- Final chaining variable is the hash value

Key || Launch missile at target A

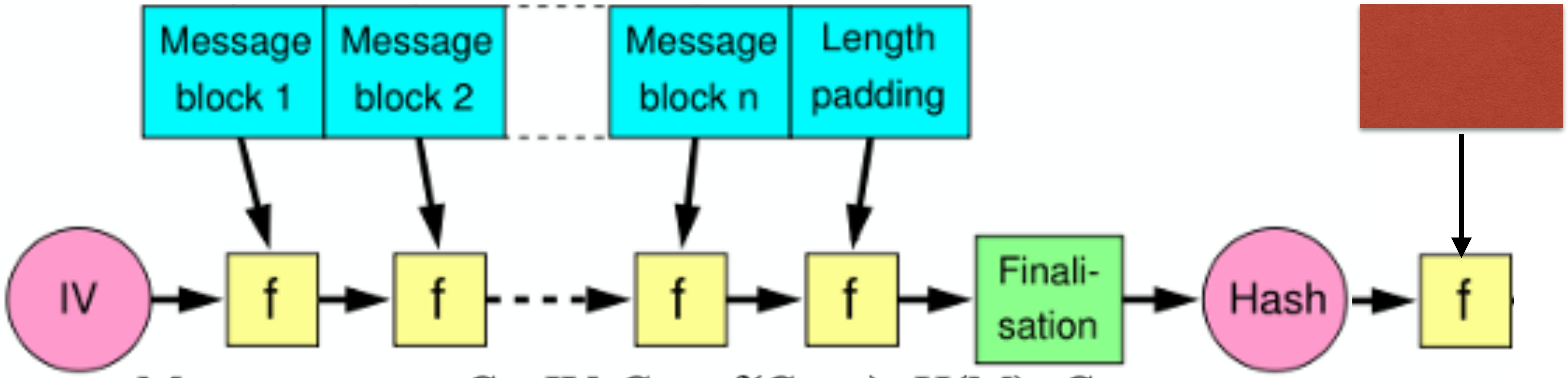


$M=m_1m_2\ldots m_n$; $C_0=IV$, $C_{i+1}=f(C_i,m_i)$; $H(M)=C_n$

Hash Length Extension Attack

- Message is divided into fixed-size blocks and padded
- Uses a compression function f, which takes a chaining variable (of size of hash output) and a message block, and outputs the next chaining variable
- Final chaining variable is the hash value

Key || Launch missile at target A and PSU



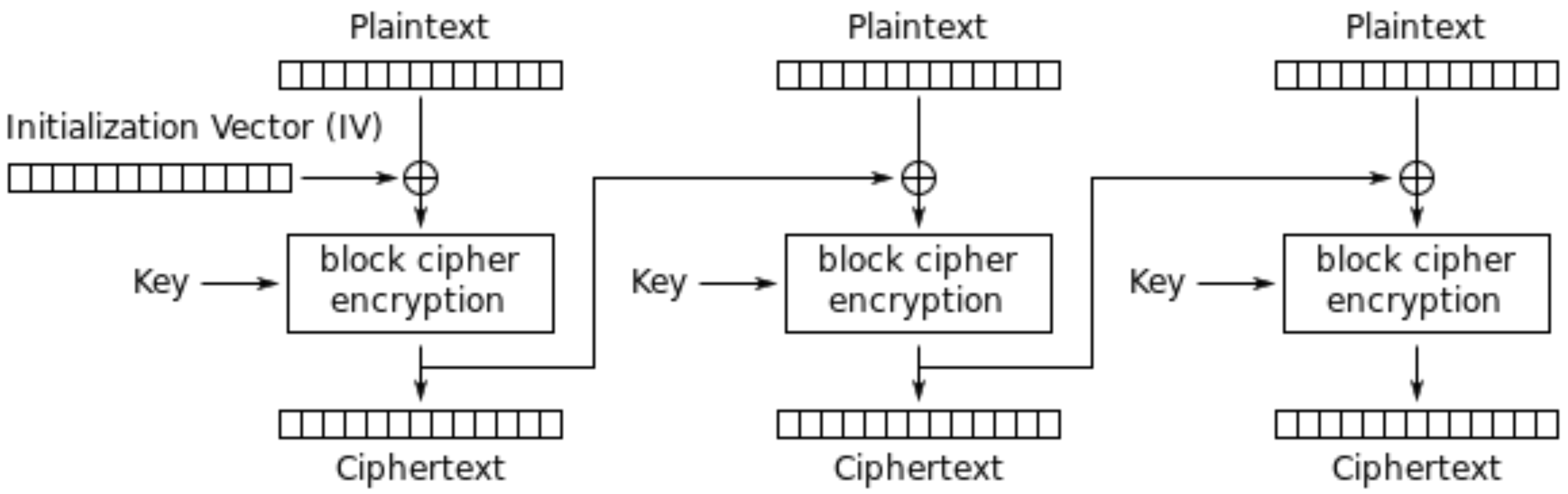
$M=m_1m_2\ldots m_n;\ C_0=IV,\ C_{i+1}=f(C_i,m_i);\ H(M)=C_n$

HMAC

- MAC that meets the following properties
  - Collision-resistant
  - Attacker cannot compute a proper digest without knowing K
    - Even if attacker can see an arbitrary number of digests H(k+x)
- Simple MAC has a flaw
  - Block hash algorithms mean that new content can be added
  - Turn H(K+m) to H(K+m+m') where m' is controlled by an attacker
- HMAC(K,m) = H(K + H(K + m))
  - Attacker cannot extend MAC as above
  - Prove it to yourself

CBC-MAC

- You can also produce a MAC using a symmetric encryption function in CBC mode
  - Encryption in CBC produces ciphertext that is dependent on all prior plaintext blocks
  - Last block of ciphertext is suitable as a MAC
    - Use different key than for encryption



Cipher Block Chaining (CBC) mode encryption

Authenticated Encryption

- Several modes of operation provide both encryption and authentication at the same time!
- Popular modes include
  - CCM (Counter with CBC-MAC)
  - GCM (Galois/Counter Mode)

Using Crypto

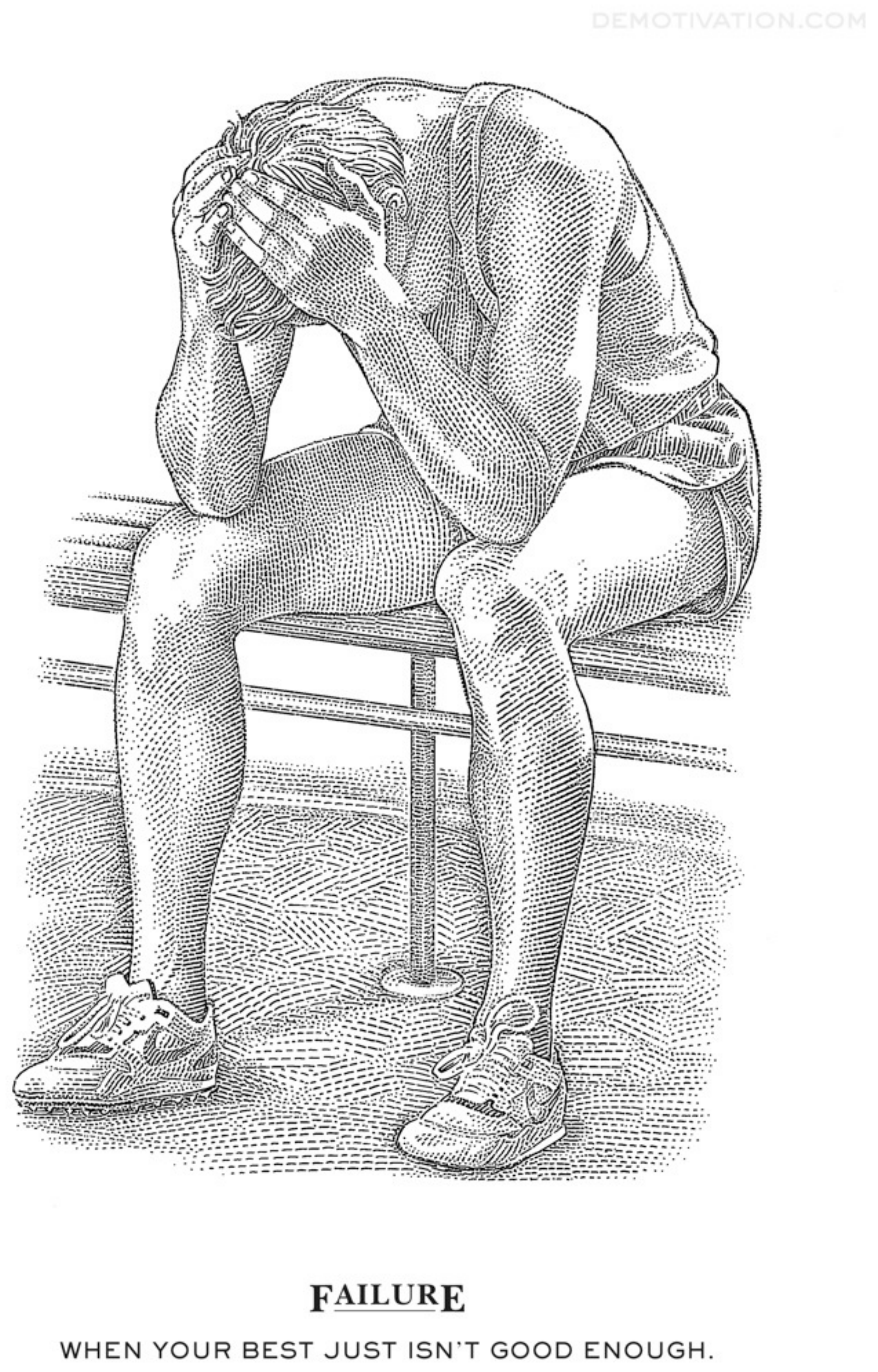
- Suppose you (Alice) want to send a document securely to another party (Bob)
  - You have each obtained a secret key
  - Obtained in some secure fashion (key distribution, later)
- How do you send the document such that only Bob can read it?
- How do you send the document such that Bob knows it is from Alice?

Basic truths of cryptography …

- Cryptography is not frequently the source of security problems
  - Algorithms are well known and widely studied
    - Use of crypto commonly is … (e.g., WEP)
  - Vetted through crypto community
  - Avoid any "proprietary" encryption
  - Claims of "new technology" or "perfect security" are almost assuredly snake oil

Why Cryptosystems Fail

- In practice, what are the causes of cryptosystem failures
  - Not crypto algorithms typically



Products Have Problems

- Despite well understood crypto foundations, products don't always work securely
  - Leak secrets due to encryption in software
  - Incompatibilities (borrow my terminal)
  - Poor product design
    - Backdoors enabled, non-standard crypto, lack of entropy, etc.
  - Sloppy operations
    - Ignore attack attempts, share keys, procedures are not defined or followed
  - Cryptanalysis sometimes
    - Home-grown algorithms!, improper parameters, cracking DES

Problems

- Systems may work in the lab/theory, but
  - Are difficult to use in practice
  - Counter-intuitive
  - Rewards aren't clear
  - Correct usage is not clear
  - Too many secrets ultimately

- Fundamentally, two problems
  - Too complex to use
  - No way to determine if use is correct

What Can We Do?

- Anderson suggests
  - Determine exactly what can go wrong
    - Find all possible failure modes
  - Put in safeguards
    - Describe how preventions protect system
  - Correct implementation of safeguards
    - Implementation of preventions meets requirements
  - Decisions left to people are small in number and clearly understood
    - People know what to do

- Problems of security in general

Building systems with cryptography

- Use quality libraries
  - E.g., OpenSSL, Libgcrypt, Cryptlib, BouncyCastle
  - Find out what cryptographers think of a package
- Code review like crazy
- Educate yourself on how to use libraries
  - Caveats by original designer and programmer

Common issues that lead to pitfalls

- Generating randomness
- Storage of secret keys
- Virtual memory (pages secrets onto disk)
- Protocol interactions
- Poor user interface
- Poor choice of key length, prime length, using parameters from one algorithm in another