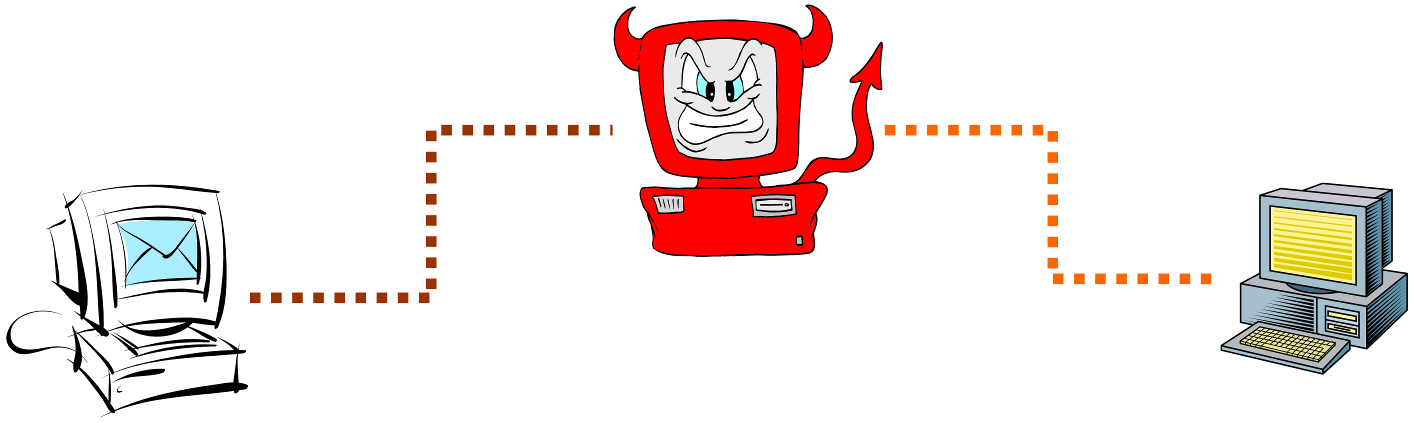
NETWORK SECURITY PRACTICES – ATTACK AND DEFENSE

Message Authentication Code

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.

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

Security Requirements for Cryptographic Hash Functions

Given a function $h: X \to Y$, 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. $x' \neq x$ 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. $h(x') = h(x)$

Uses of hash functions

- Message authentication
- Software integrity
- One-time Passwords
- Digital signature
- Timestamping

Brute-force Attacks on Hash Functions

- Attacking one-wayness
  - Goal: given $h: X \rightarrow Y$, $y \in Y$, find x such that $h(x)=y$
  - Algorithm:
    - pick a random value x in X, check if $h(x)=y$, if $h(x)=y$, returns x; otherwise iterate
    - after failing q iterations, return fail
  - The average-case success probability is

$$\varepsilon = 1 - \left(1 - \frac{1}{|Y|}\right)^q \approx \frac{q}{|Y|}$$

  - Let $|Y|=2^m$, to get ε to be close to 0.5, $q \approx 2^{m-1}$

Brute-force Attacks on Hash Functions

- Attacking collision resistance
  - Goal: given h, find x, x’ such that h(x)=h(x’)
  - Algorithm:
    - pick a random set X_0 of q values in X for each $x \in X_0$
    - Computes $y_x = h(x)$
    - If $y_x = y_{x'}$ for some $x' \neq x$ then return (x,x’) else fail
  - The average success probability is $$1 - e^{-\frac{q(q-1)}{2|Y|}}$$
  - Let $|Y| = 2^m$, to get ε to be close to 0.5, $q \approx 2^{m/2}$
  - This is known as the birthday attack.

Well Known Hash Functions

- MD5
  - Output 128 bits
  - Not collision resistance
    - http://merlot.usc.edu/csac-f06/papers/Wang05a.pdf
- SHA1
  - output 160 bits
  - no collision found yet, but method exist to find collisions in less than 2^{63} (Xiaoyun Wang, Andrew Yao and Frances Yao)
  - considered insecure for collision resistance
  - one-wayness
- SHA-256, SHA-384, SHA-512
  - outputs 256, 384, and 512 bits, respectively
- NIST is requesting submissions of new standard hash algorithms

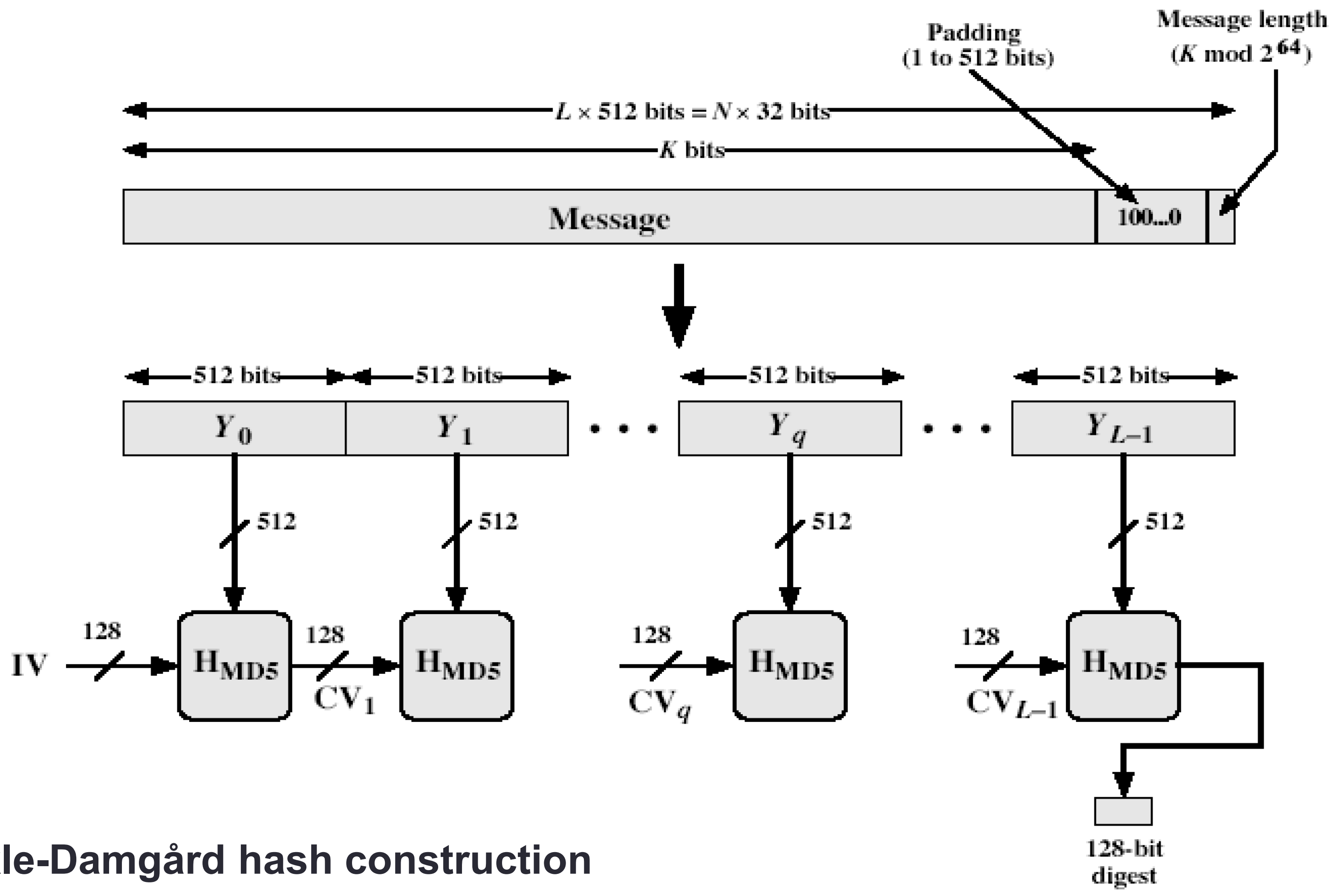
Choosing the length of Hash outputs

- Because of the birthday attack, the length of hash outputs in general should double the key length of block ciphers
  - SHA-256, SHA-384, SHA-512 to match the new key lengths (128,192,256) in AES

Iterative Construction of Hash Functions

- A hash function needs to map a message of an arbitrary length to a m-bit output
  - h: $\{0,1\}^* \rightarrow \{0,1\}^m$
- The iterative construction
  - use a compression function that takes a fixed-length input string and output a shorter string
    - f:$\{0,1\}^{m+t} \rightarrow \{0,1\}^m$
  - A message is divided into fixed length blocks and processed block by block

Iterative Construction of MD5



Merkle-Damgård hash construction

Limitation of Using Hash Functions for Authentication

- Require an authentic channel to transmit the hash of a message
  - anyone can compute the hash value of a message, as the hash function is public
  - not always possible
- How to address this?
  - use more than one hash functions
  - use a key to select which one to use

Hash Family

- A hash family is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, where
  - $\mathcal{X}$ is a set of possible messages
  - $\mathcal{Y}$ is a finite set of possible message digests
  - $\mathcal{K}$ is the keyspace
  - For each $K \in \mathcal{K}$, there is a hash function $h_K \in \mathcal{H}$. Each h_K: $\mathcal{X} \rightarrow \mathcal{Y}$
- Alternatively, one can think of $\mathcal{H}$ as a function $\mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$

Message Authentication Code

- A MAC scheme is a hash family, used for message authentication
- MAC = $h_K(M)$
- The sender and the receiver share K
- The sender sends (M, $h_k(M)$)
- The receiver receives (X,Y) and verifies that $h_K(X)=Y$, if so, then accepts the message as from the sender
- To be secure, an adversary shouldn't be able to come up with (X,Y) such that $h_K(X)=Y$.

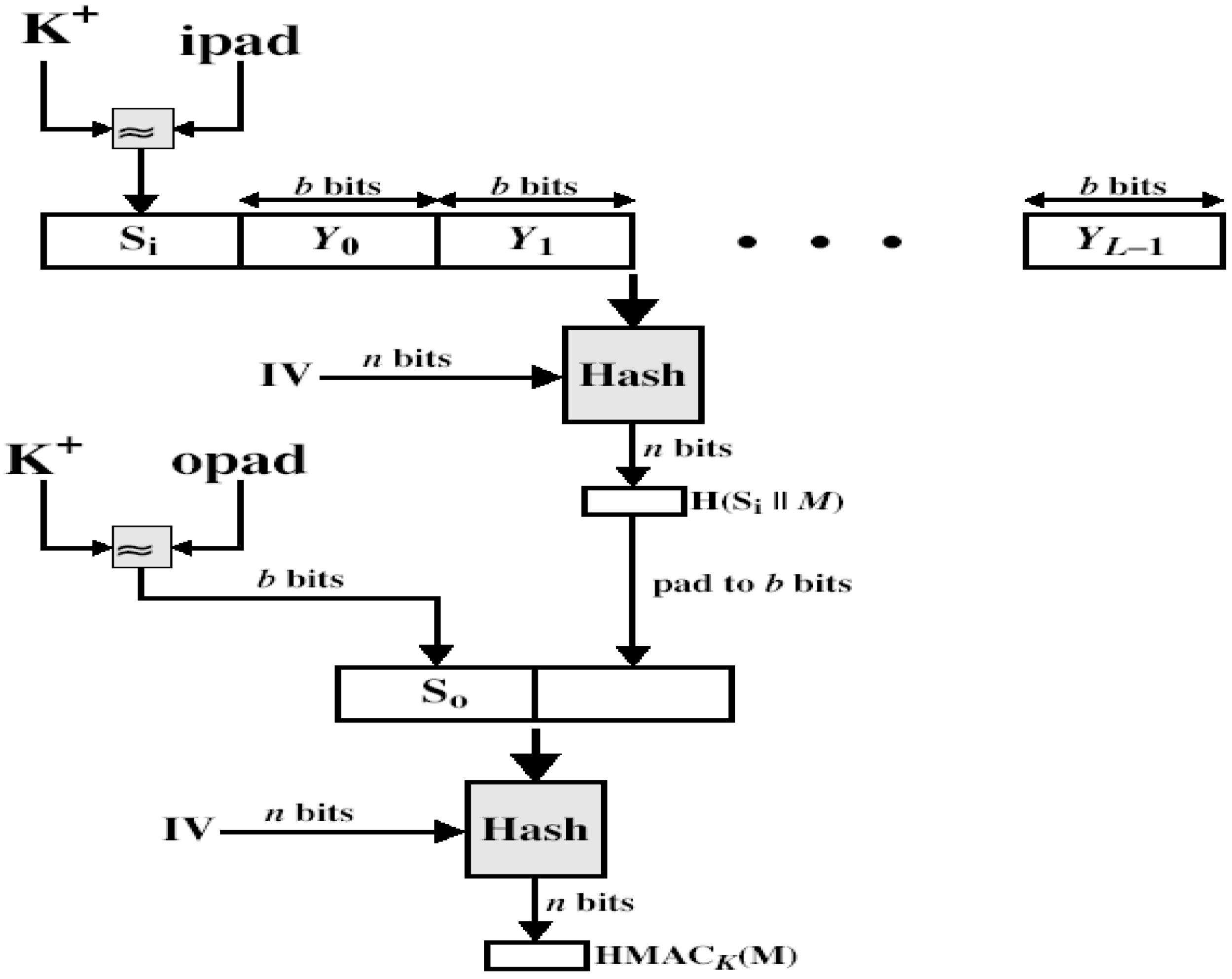
HMAC: Constructing MAC from Cryptographic Hash Functions

$\text{HMAC}_K[M] = \text{Hash}[(K^+ \oplus \text{opad}) \,||\, \text{Hash}[(K^+ \oplus \text{ipad})||M)]]$

- K^+ is the key padded (with 0) to B bytes, the input block size of the hash function
- ipad = the byte 0x36 repeated B times
- opad = the byte 0x5C repeated B times.

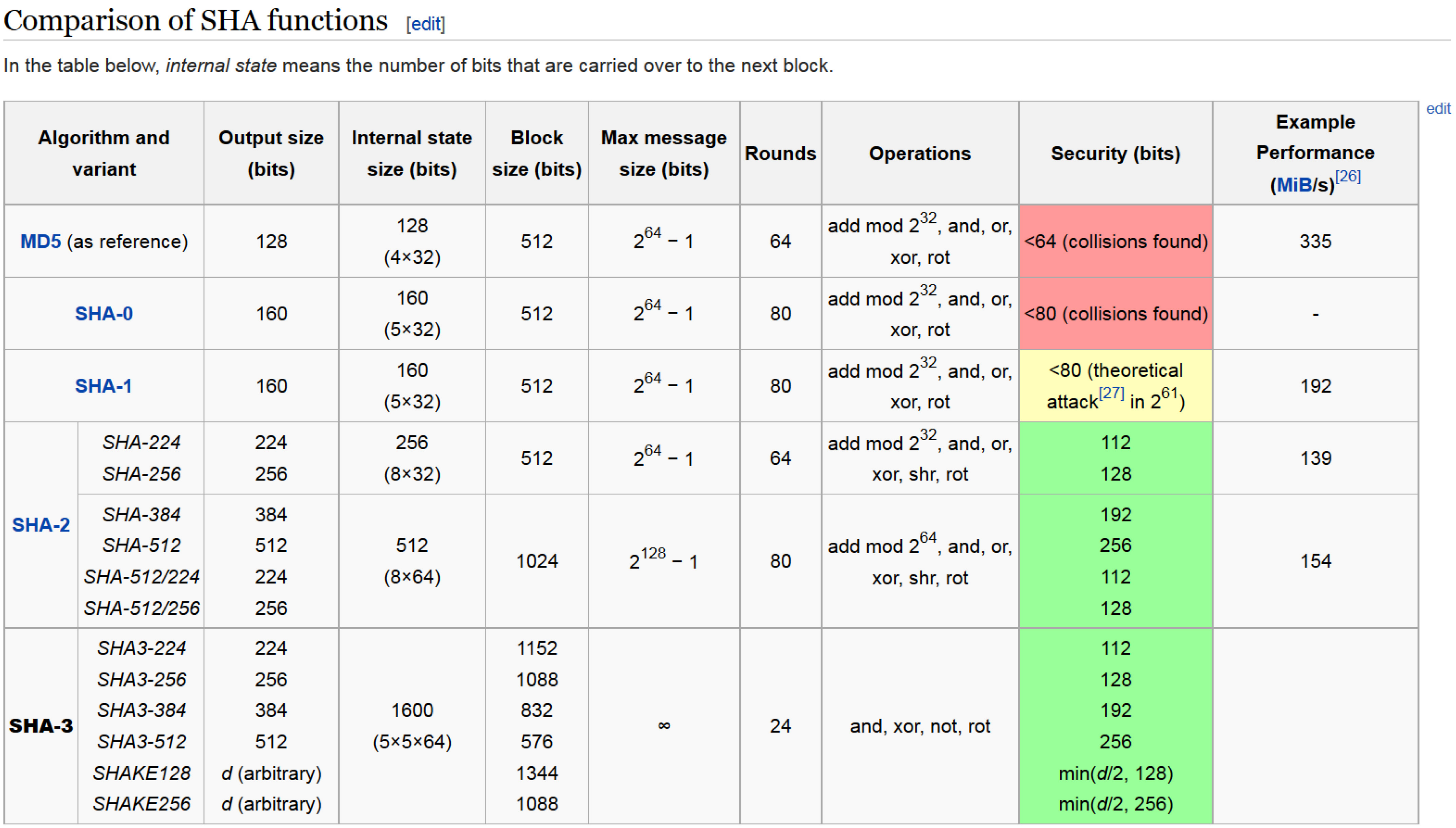
http://cseweb.ucsd.edu/~mihir/papers/hmac.html

HMAC Overview



HMAC Security

- If used with a secure hash functions (e.g., SHA-256) and according to the specification (key size, and use correct output), no known practical attacks against HMAC



Comparison of SHA functions [edit]

In the table below, *internal state* means the number of bits that are carried over to the next block.

edit

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Rounds	Operations	Security (bits)	Example Performance (MiB/s)[26]
MD5 (as reference)		128	128 (4×32)	512	$2^{64} - 1$	64	add mod 2^{32}, and, or, xor, rot	<64 (collisions found)	335
SHA-0		160	160 (5×32)	512	$2^{64} - 1$	80	add mod 2^{32}, and, or, xor, rot	<80 (collisions found)	-
SHA-1		160	160 (5×32)	512	$2^{64} - 1$	80	add mod 2^{32}, and, or, xor, rot	<80 (theoretical attack[27] in 2^{61})	192
SHA-2	*SHA-224* *SHA-256*	224 256	256 (8×32)	512	$2^{64} - 1$	64	add mod 2^{32}, and, or, xor, shr, rot	112 128	139
	SHA-384 *SHA-512* *SHA-512/224* *SHA-512/256*	384 512 224 256	512 (8×64)	1024	$2^{128} - 1$	80	add mod 2^{64}, and, or, xor, shr, rot	192 256 112 128	154
SHA-3	*SHA3-224* *SHA3-256* *SHA3-384* *SHA3-512* *SHAKE128* *SHAKE256*	224 256 384 512 *d* (arbitrary) *d* (arbitrary)	1600 (5×5×64)	1152 1088 832 576 1344 1088	∞	24	and, xor, not, rot	112 128 192 256 min(*d*/2, 128) min(*d*/2, 256)	

Encryption and Authentication

- Three ways for encryption and authentication
  - Authenticate-then-encrypt (AtE), used in SSL
    - a = MAC(x), C=E(x,a), transmit C
  - Encrypt-then-authenticate (EtA), used in IPSec
    - C=E(x), a=MAC(C), transmit (C,a)
  - Encrypt-and-authenticate (E&A), used in SSH
    - C=E(x), a=MAC(x), transmit (C,a)

Encryption and Authentication

- Which way provides secure communications when embedded in a protocol that runs in a real adversarial network setting?
  - "The Order of Encryption and Authentication for Protecting Communications, " Hugo Krawzyck.
  - "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm, " M. Bellare and C. Namprempre
  - "Reconsidering Generic Composition," C. Namprempre et al.

“Secure” Channel

- Requirement of “secure” channel
  - any message accepted as valid for a session by a receiving party was indeed sent by the partner to the session
  - attacker cannot learn any information about messages
  - even against an adversary that fully controls the communication channel and may learn information through responses of the participants

- Encrypt-then-authenticate (EtA) provides “secure” channel
  - C=E(x), a=MAC(C), transmit (C,a)

Using Only Encryption Schemes is Insufficient

- Given a secure stream cipher (or even one-time pad) E, Consider encryption E*
    - E*[x] = E[encode[x]]
        - encode[x] replaces each bit 0 with 00, and bit 1 with either 01 or 10.
    - How to decrypt?

- Using E* does not provide confidentiality in a real world protocol setting
    - if adversary flips the first two bits of E*[x] and can learn whether decryption succeeds, then the adversary learns the first bit of x

AtE and E&A are insecure

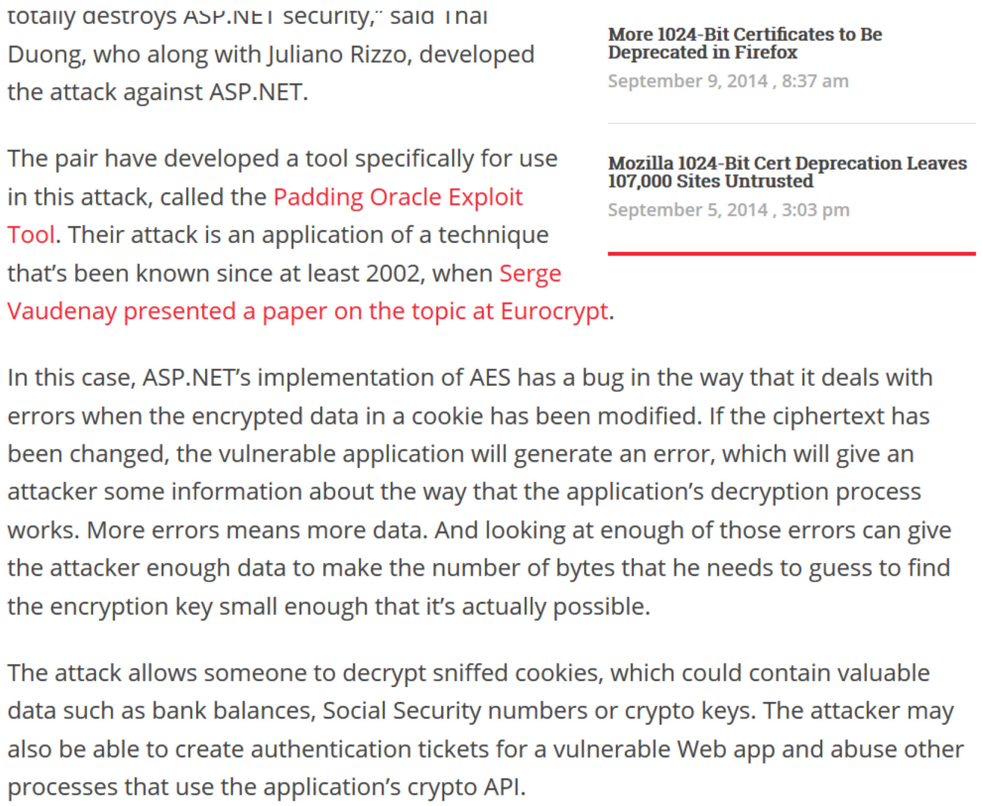
- Authenticate-then-encrypt (AtE) is not secure in general
  - a = MAC(x), C=E(x,a), transmit C
  - consider flipping the first two bits of C
    - if first bit of x is 0, decryption fails
    - if first bit of x is 1, decryption & MAC verification succeeds.
  - AtE, however, can be secure for some specific encryption schemes, such as CBC or OTP (or stream ciphers)

- Encrypt-and-authenticate (E&A) is not secure in general
  - C=E(x), a=MAC(x), transmit (C,a)
  - Why?

Padding Oracle Attack

- Serge Vaudenay (2002). "Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS...". EUROCRYPT 2002.
- Juliano Rizzo, Thai Duong (2010-05-25). "Practical Padding Oracle Attacks". USENIX WOOT 2010.
- https://blog.skullsecurity.org/2013/a-padding-oracle-example

- PadBuster
  - http://blog.gdssecurity.com/labs/2010/9/14/automated-padding-oracle-attacks-with-padbuster.html



totally destroys ASP.NET security," said Thai Duong, who along with Juliano Rizzo, developed the attack against ASP.NET.

The pair have developed a tool specifically for use in this attack, called the Padding Oracle Exploit Tool. Their attack is an application of a technique that's been known since at least 2002, when Serge Vaudenay presented a paper on the topic at Eurocrypt.

In this case, ASP.NET's implementation of AES has a bug in the way that it deals with errors when the encrypted data in a cookie has been modified. If the ciphertext has been changed, the vulnerable application will generate an error, which will give an attacker some information about the way that the application's decryption process works. More errors means more data. And looking at enough of those errors can give the attacker enough data to make the number of bytes that he needs to guess to find the encryption key small enough that it's actually possible.

The attack allows someone to decrypt sniffed cookies, which could contain valuable data such as bank balances, Social Security numbers or crypto keys. The attacker may also be able to create authentication tickets for a vulnerable Web app and abuse other processes that use the application's crypto API.

More 1024-Bit Certificates to Be Deprecated in Firefox
September 9, 2014 , 8:37 am

Mozilla 1024-Bit Cert Deprecation Leaves 107,000 Sites Untrusted
September 5, 2014 , 3:03 pm

http://threatpost.com/padding-oracle-crypto-attack-affects-millions-aspnet-apps-091310/74457

Lucky Thirteen Attack

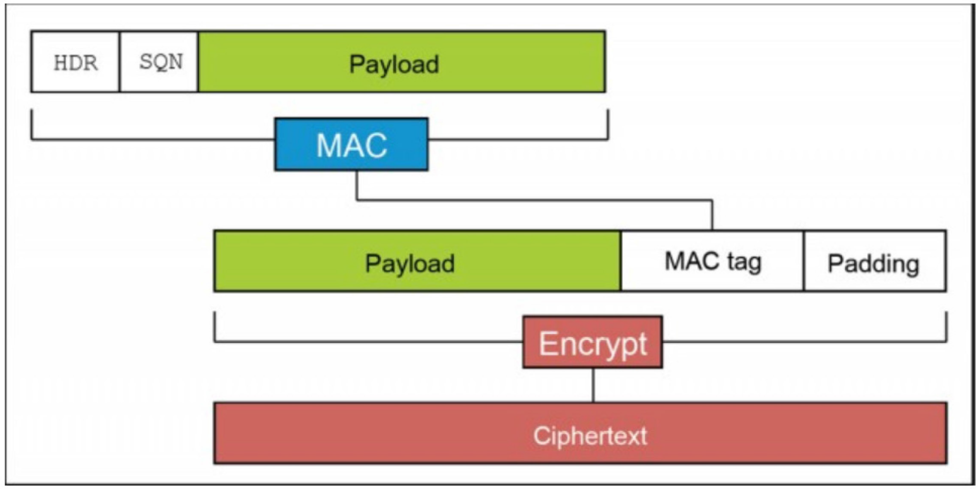
RISK ASSESSMENT / SECURITY & HACKTIVISM

“Lucky Thirteen” attack snarfs cookies protected by SSL encryption

Exploit is the latest to subvert crypto used to secure Web transactions.

by **Dan Goodin** - Feb 4 2013, 10:14pm +0800

Share Tweet 18



Enlarge / A representation of how TLS works.

Nadhem J. AlFardan and Kenneth G. Paterson

http://arstechnica.com/security/2013/02/lucky-thirteen-attack-snarfs-cookies-protected-by-ssl-encryption/