Introduction to Cryptography

Cryptography Introduction/Refresher

- Brief introduction to make sure everyone's is on the same page
- Important concepts:
  - Symmetric ciphers
  - Public key encryption
  - Digital signatures
  - Cryptographic hash functions
  - Message Authentication Codes (MACs)
  - Certificates

What is a Cryptosystem?

- $K = \{0,1\}^l$
- $P = \{0,1\}^m$
- $C = \{0,1\}^m$
- $E : P \times K \rightarrow C$
- $D : C \times K \rightarrow P$
- $\forall p \in P, k \in K : D(E(p,k),k) = p$
- It is infeasible to find $F : P \times C \rightarrow K$

Let's start again, in English. . .

What is a Cryptosystem?

A cryptosystem is pair of algorithms that take a *key* and under control of that key convert *plaintext* to *ciphertext* and back.

Plaintext is what you want to protect; ciphertext should appear to be random gibberish.

The design and analysis of today's cryptographic algorithms is highly mathematical. Do *not* try to design your own algorithms.

Properties of a Good Cryptosystem

- There should be no way short of enumerating all possible keys (a "brute force" or "exhaustive search" attack) to find the key from any amount of ciphertext and plaintext, nor any way to produce plaintext from ciphertext without the key.
- Enumerating all possible keys must be infeasible.
- The ciphertext must be indistinguishable from true random values.

Kerckhoffs' Law (1883)

> There must be no need to keep the system secret, and it must be able to fall into enemy hands without inconvenience.

In other words, the security of the system must rest entirely on the secrecy of the key.

Keys

- Must be strongly protected
- Ideally, should be a random set of bits of the appropriate length
- Ideally, each key should be only be used for a limited time and for a limited amount of data
- Ensuring that these properties hold is a major goal of cryptographic research and engineering

Cipher Strengths

- A cipher is no stronger than its key length: if there are too few keys, an attacker can enumerate all possible keys
- The old DES cipher has 56 bit keys — arguably too few in 1976; far too few today. (*Deep Crack* was built in 1996 by the EFF for \$250,000; today's equivalent costs \$5,000 and is faster.)
- Strength of cipher depends on how long it needs to resist attack.
- No good reason to use less than 128-bit keys
- NSA rates 128-bit AES as good enough for SECRET traffic; 256-bit AES is good enough for TOP-SECRET traffic.
- But a cipher can be considerably weaker! (A monoalphabetic cipher over all possible byte values has 256! keys — a length of 1684 bits — but is trivially solvable.)

Brute-Force Attacks

- Build massively parallel machine
- Can be distributed across the Internet
- Give each processor a set of keys and a plaintext/ciphertext pair
- If no known plaintext, look for probable plaintext (i.e., length fields, high-order bits of ASCII text, etc.)
- On probable hit, check another block and/or do more expensive tests

Expensive Tests

Which is real language? How can you tell?

tqxxa iadxp	spwwz hzcwo
rovvy gybvn	jgnnq yqtnf
ifmmp xpsme	hello world
gdkkn vnqkc	fcjjm umpjb
erqmrxu oh prqgh	dqplqwt ng oqpfg
cpokpvs mf npoef	bonjour le monde
anmintq kd lnmcd	adsxika ak gadfk

Some approaches: the bigraph distribution, letter contact probabilities, entropy of the message, etc.

CPU Speed versus Key Size

- Adding one bit to the key doubles the work factor for brute force attacks
- The effect on encryption time is often negligible or even free
- It costs *nothing* to use a longer RC4 key
- Going from 128-bit AES to 256-bit AES takes (at most) 40% longer, but increases the attacker's effort by a factor of 2^{128}
- Using triple DES costs $3\times$ more than DES to encrypt, but increases the attacker's effort by a factor of 2^{112}
- Moore's Law favors the defender

Block Ciphers

- Operate on a fixed-length set of bits
- Output blocksize generally the same as input blocksize
- Well-known examples: DES (56-bit keys; 64-bit blocksize); 3DES (112-bit keys; 64-bit blocksize); AES (128-, 192-, and 256-bit keys; 128-bit blocksize)

Stream Ciphers

- Key stream generator produces a sequence S of pseudo-random bytes; key stream bytes are combined (generally via XOR) with plaintext bytes: $P_i \oplus S_i \rightarrow C_i$
- Stream ciphers are very good for asynchronous traffic
- Best-known stream cipher is RC4; commonly used with TLS. (RC4 is now considered insecure; don't use it for new deployments and get rid of existing uses.)
- Key stream S must *never* be reused for different plaintexts:

$$\begin{aligned} C &= A \oplus K \\ C' &= B \oplus K \\ C \oplus C' &= A \oplus K \oplus B \oplus K \\ &= A \oplus B \end{aligned}$$

- Guess at A and see if B makes sense; repeat for subsequent bytes

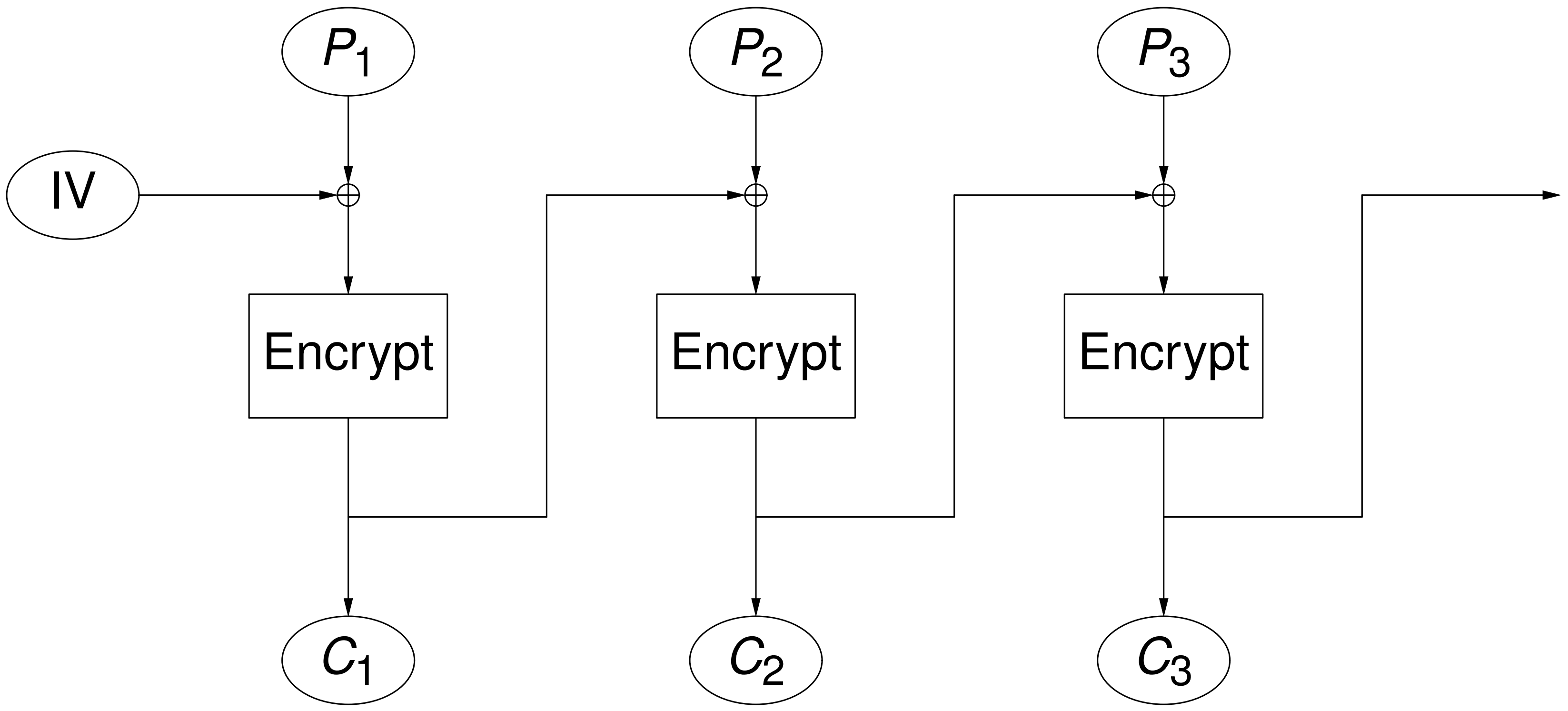
Basic Structure of (Most) Block Ciphers

- Optional key scheduling — convert supplied key to internal form
- Multiple *rounds* of combining the plaintext with the key.
- DES has 16 rounds; AES has 10–14 rounds, depending on key length

Modes of Operation

- Direct use of a block cipher is inadvisable
- Enemy can build up “code book” of plaintext/ciphertext equivalents
- Beyond that, direct use only works on messages that are a multiple of the cipher block size in length
- Solution: several standard *Modes of Operation*, including Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), Galois Counter Mode (GCM), Counter (CTR), and more. (Some modes provide authentication as well as confidentiality.)
- All modes of operation except ECB require an extra block known as the *Initialization Vector* (IV). IVs *must* be unpredictable by the enemy.

Example: Cipher Block Chaining

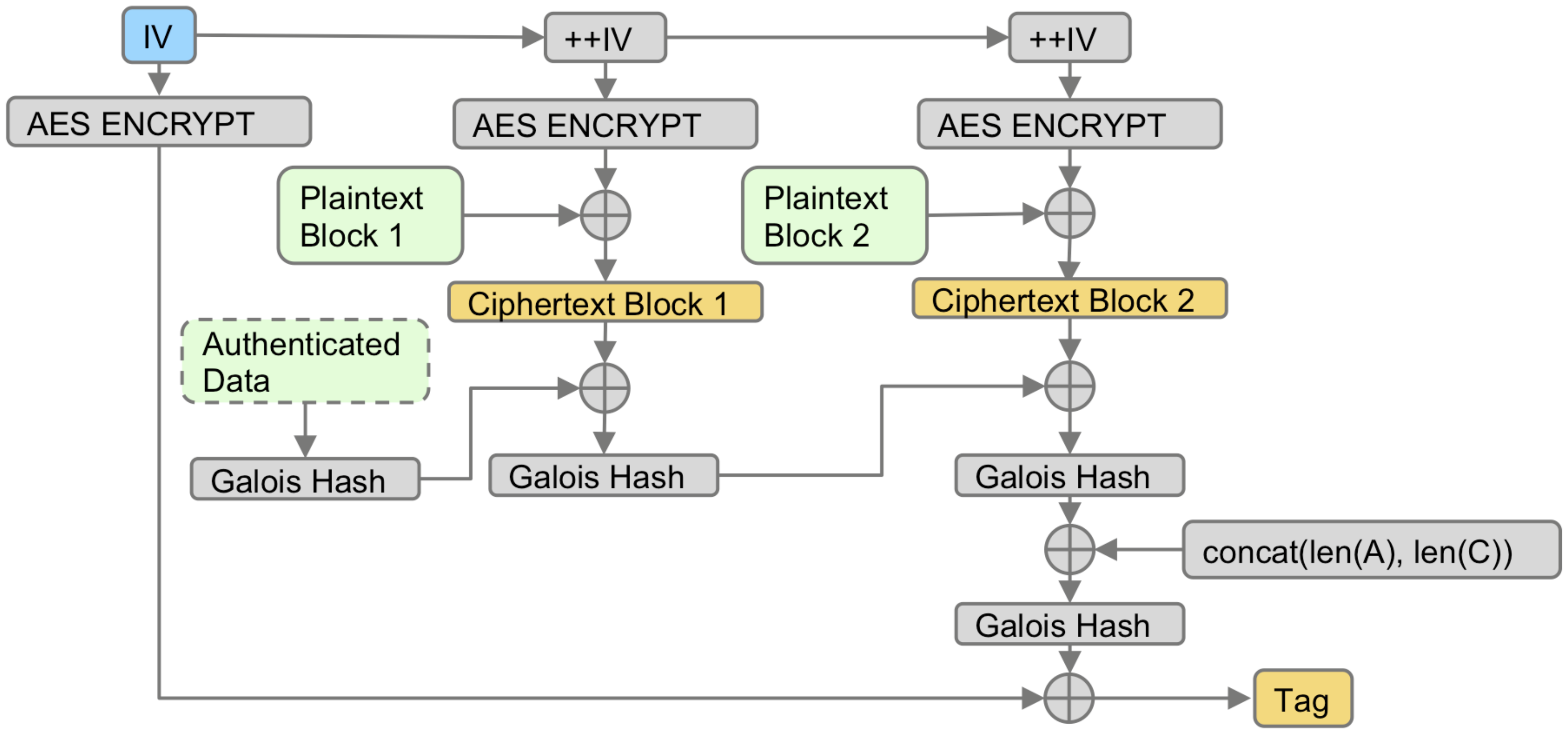


$$\{P_i \oplus C_{i-1}\}_k \rightarrow C_i$$
$$\{C_i\}_{k^{-1}} \oplus C_{i-1} \rightarrow P_i$$

Things to Notice About CBC

- Identical plaintext blocks do not, in general, produce the same ciphertext. (Why?)
- Each ciphertext block is a function of all previous plaintext blocks. (Why?)
- The converse is not true, but we won't go into that in this class
- It is possible for an attacker to make *partially controlled* changes to plaintext, by tinkering with the previous block of ciphertext
- We need *message authentication* in addition to encryption

Galois Counter Mode Encryption



Galois Counter Mode

- Much more complicated!
- Encrypts and authenticates in one pass
- Part of the message can be sent in plaintext, but is still authenticated
- Output is ciphertext plus a “tag”
- Encryptions can be done in parallel (good for high-speed devices)—the IV is a counter
- If the tag computed during decryption doesn’t match the tag computed during encryption, the decryption routine returns “FAIL”

Alice and Bob

- Alice wants to communicate securely with Bob
- (Cryptographers frequently speak of Alice and Bob instead of *A* and *B*. . .)
- What key should she use?

Pre-Arranged Key Lists?

- What if you run out of keys?
- What if a key is stolen?

  “Why is it necessary to destroy yesterday’s [key] . . . list if it’s never going to be used again?”

  “A used key, Your Honor, is the most critical key there is. If anyone can gain access to that, they can read your communications.”

  (trial of Jerry Whitworth, a convicted spy.)
- What if Alice doesn’t know in advance that she’ll want to talk to Bob?

The Solution: Public Key Cryptography

- Allows parties to communicate without prearrangement
- Separate keys for encryption and decryption
- Not possible to derive decryption key from encryption key
- Permissible to publish encryption key, so that anyone can send you secret messages
- All known public key systems are very expensive to use, in CPU time and bandwidth.
- Most public systems are based on mathematical problems.

RSA

- The best-known public key system is RSA.
- Generate two large (these days, at least 1024-bit) primes p and q; let $n = pq$
- Pick two integers e and d such that $ed \equiv 1 \bmod (p-1)(q-1)$. Often, $e = 65537$, since that simplifies encryption calculations. (Older systems use $e = 3$, but that's no longer recommended.)
- The public key is $\langle e, n \rangle$; the private key is $\langle d, n \rangle$.
- To encrypt m, calculate $c = m^e \bmod n$; to decrypt c, calculate $m = c^d \bmod n$.
- The security of the system (probably) relies on the difficulty of factoring n.
- Finding such primes is relatively easy; factoring n is believed to be extremely hard.

Classical Public Key Usage

- Alice publishes her public key in the phone book.
- Bob prepares a message and encrypts it with that key by doing a large exponentiation.
- Alice uses her private key to do a different large exponentiation.
- It’s not that simple—more in a few minutes. . .

Complexities

- RSA calculations are *very* expensive; neither Bob nor Alice can afford to do many.
- RSA is too amenable to mathematical attacks; encrypting the wrong numbers is a bad idea.
- Example: “yes”^{3} is only 69 bits, and won’t be reduced by the modulus operation; finding $\sqrt[3]{503565527901556194283}$ is easy.
- We need a better solution

Speeds on my Laptop

Algorithm	*ops/second*	*bytes/sec*
AES-128-ECB	9,250,000	148,000,000
AES-128-CBC (256 bytes)	600,000	153,000,000
AES-256-ECB	7,140,000	114,240,000
RSA-2048 (encrypt)	14,900	3,800,000
RSA-2048 (decrypt)	375	96,000

A (More) Realistic Scenario

- Bob generates a random key k for a conventional cipher.
- Bob encrypts the message: $c = \{m\}_k$.
- Bob pads k with a known amount of padding, to make it at least 1024 bits long; call this k'.
- k' is encrypted with Alice's public key $\langle e, n\rangle$.
- Bob transmits $\{c, (k')^e \bmod n\}$ to Alice.
- Alice uses $\langle d, n\rangle$ to recover k', removes the padding, and uses k to decrypt ciphertext c.
- In reality, it's even more complex than that...

Forward Secrecy

- Someone who learns your private key can read old messages.
- Solution: Diffie-Hellman Key Exchange:

$$A \rightarrow B : g^x \bmod p$$
$$B \rightarrow A : g^y \bmod p$$

  where p is a large prime, b is a generator of the group of integers modulo p, and x and y are random integers $< p$.
- A and B can calculate $g^{xy} \bmod p$; no eavesdropper can without solving the *discrete log* problem.
- Use that number as the data key—not recoverable later if x and y are destroyed. This is called *forward secrecy*.

Who Sent a Message?

- When Bob receives a message from Alice, how does he know who sent it?
- With traditional, symmetric ciphers, he may know that Alice has the only other copy of the key; with public key, he doesn't even know that
- Even if he knows, can he prove to a third party — say, a judge — that Alice sent a particular message?

Digital Signatures

- RSA can be used backwards: you can encrypt with the private key, and decrypt with the public key.
- This is a *digital signature*: only Alice can sign her messages, but anyone can verify that the message came from Alice, by using her public key
- It's too expensive to sign the whole message. Instead, Alice calculates a *cryptographic hash* of the message and signs the hash value.
- If you sign the plaintext and encrypt the signature, the signer's identity is concealed; if you sign the ciphertext, a gateway can verify the signature without having to decrypt the message.

They’re Not Like Real Signatures

- Real signatures are strongly bound to the person, and weakly bound to the data
- Digital signatures are strongly bound to the data, and weakly bound to the person — what if the key is stolen (or deliberately leaked)?
- A better term: digital signature algorithms provide *non-repudiation*

Cryptographic Hash Functions

- Produce relatively-short, fixed-length output string from arbitrarily long input.
- Computationally infeasible to find two different input strings that hash to the same value (“collision”)
- Computationally infeasible to find any input string that hashes to a given value (“pre-image”)
- Computationally infeasible to find any input string that hashes to the same value as the hash of a given input (“second preimage”)
- Strength roughly equal to half the output length
- 128 bits and shorter are not very secure for general usage

Common Hash Functions

- Best-known cryptographic hash functions: MD5 (128 bits), SHA-1 (160 bits), SHA2-256/384/512 (256/384/512 bits)
- Wang et al. have found collision attacks against MD5 and SHA-1
  ☞ They're insecure. MD5 is basically gone; SHA-1 is rapidly being phased out—browsers do not accept certificates signed with SHA-1 after January 1, 2016
- SHA2-256/384/512 have the same basic structure as MD5 and SHA-1—but NIST now believes they're secure
- NIST held a design competition for a new hash SHA-3 function; the winner (Keccak) has a completely different structure

The Birthday Paradox

- How many people need to be in a room for the probability that two will have the same birthday to be $> .5$?
- Naive answer: 183
- Correct answer: 23
- The question is not “who has the same birthday as Alice?”; it’s “who has the same birthday as Alice or Bob or Carol or . . . ” assuming that none of them have the same birthday as any of the others

The Birthday Attack

- Alice can prepare lots of variant contracts, looking for any two that have the same hash
- More precisely, she generates many trivial variants on m and m', looking for a match between the two sets
- This is much easier than finding a contract that has the same hash as a given other contract
- As a consequence, the strength of a hash function against brute force attacks is approximately half the output block size: 64 bits for MD5, 80 bits for SHA-1, etc.

Message Integrity

- We need a way to prevent tampering with messages
- One choice: use GCM
- Or—we can use a key and a cryptographic hash to generate a *Message Authentication Code* (MAC).
- One bad idea: append a cryptographic hash to some plaintext, and encrypt the whole thing with, say, CBC mode

$$\{P, H(P)\}_K$$

- This can fall victim to a *chosen plaintext attack*

HMAC

- Build a MAC from a cryptographic hash function
- Best-known construct is HMAC — provably secure under minimal assumptions
- HMAC$(m, k) = H(\text{opad} \oplus k, H(\text{ipad} \oplus k, m))$ where H is a cryptographic hash function and m is the message
- Note: do the HMAC over the *ciphertext*, not the plaintext
- Note: authentication key *must* be distinct from the confidentiality key
- Frequently, the output of HMAC is truncated

Cryptography and Authentication

- Some way to use a cryptographic key to prove who you are
- (More on that later)
- Can go beyond simple schemes given above
- Can use symmetric or public key schemes
- Most public key schemes use *certificates*

What are Certificates

- How does Alice get Bob’s public key?
- What if the enemy tampers with the phone book? Sends the phone company a false change-of-key notice? Interferes with Alice’s query to the phone book server?
- A certificate is a digitally-signed message containing an identity and a public key — prevents tampering.

Why Trust a Certificate?

- Who signed it? Why do you trust them?
- How do you know the public key of the *Certificate Authority* (CA)?
- Some public key (known as the *trust anchor*) must be provided out-of-band — trust has to start somewhere.

Certificate Authorities

- Who picks CAs? No one and every one.
- Your browser has some CAs built-in — because the CA paid the browser vendor enough money. Is that grounds for trust?
- Matt Blaze: “A commercial certificate authority can be trusted to protect you from anyone from whom they won’t take money.”

What Else is in a Certificate?

- Technical information, such as algorithm identifiers
- More identification information — company, location, etc.
- Expiration date
- Logos
- Certificate role

Cryptographic Protocols

- Combine various cryptographic primitives in a series of messages
- Many different types, for many different goals
- Simplest example: “realistic” public key encryption message discussed earlier: $\langle \{m\}_k, (\text{pad}(k))^e \bmod n \rangle$
- Very common goal: Alice and Bob must agree on a key
- Very subtle; very hard to get right. Don’t try it yourself

Elliptic Curve Cryptography

- Based on points on curves of the form $y^2 = x^3 + ax + b$ — but use only integers modulo some large prime
- There are analogs to the most important public key and digital signature primitives using elliptic curves
- Keys are much shorter and much less CPU time is needed for a given level of security

Cryptography and Quantum Computers

- Quantum computers—if they can be built—can crack a symmetric cipher in time proportional to the square root of the key space size

☞ For long-term secrets, use 256-bit AES, not 128-bit

- They can also do factoring efficiently

☞ RSA won't be secure; we need quantum-safe algorithms

- Many elliptic curve algorithms are also vulnerable
- If your secrets have to be protected for decades, you have to take action now
- The NSA has called for migration to "post-quantum cryptographic algorithms"—but there's not yet a consensus on the proper public key algorithm

Recommended Primitives

- Block cipher: AES, either with HMAC or (preferably) in GCM
- Stream cipher: RC4? It's insecure and all but dead; use AES in CTR mode instead
- Hash function: SHA2-256 or SHA3
- Public key, digital signature: RSA with 2048-bit modulus (or Elliptic Curve Cryptography if patents aren't an issue and performance is)
- Stay tuned for the preferred "post-quantum" algorithm