# CRYPTOGRAPHY

EN.600.444/644

Fall 2019

**Dr. Seth James Nielson**

## THIS IS NOT A CRYPTOGRAPHY CLASS

- And your textbook is not a crypto book
  - Optional: "Handbook of Applied Cryptography" (HAC)
- We will not be discussing the mathematics
- Focus on "black box" crypto primitives
  - Hashing
  - Symmetric Operations
  - Asymmetric Operations

## SOME TERMINOLOGY

- Cryptography – How to encrypt
- Cryptanalysis – How to break ciphers
- Cryptology – Cryptography + Cryptanalysis
- Plaintext/Ciphertext – Unencrypted/Encrypted
- Ciphers – Mechanisms for encrypting (stream/block)
  - Symmetric - Shared key
  - Asymmetric - public/private key
    - Digital signatures
- Hash functions – One-way functions

## THE CRYPTOGRAPHY PROBLEM

- Cryptography underlies almost all modern computer security
- Yet, it is surprisingly hard to use correctly
- Let’s review some history

## EARLY SUBST. CIPHER

- Substitution Ciphers
  - For example, monoalphabetic substitution (Caesar cipher)
  - Easy to break; strengthen with block or stream ciphers

- Let's race! Decrypt the following:

**RYG WKXI CSLVSXQC NY IYE RKFO**

  - It's a question, when you decrypt it, shout out the answer
  - First one to decrypt it wins a prize

# EARLY STREAM CIPHER

- Vigenere Cipher

  PLAINTEXT: YOUPASSEDTHECLASS

  KEY: PUPPYPUPPYPUPPYPUPPY

  CIPHERTXT: NIJEYHMTSRWYRAYHM

- Still breakable with enough text
- Playfair – Early Block Cipher
  - Much harder to break because it’s encoding digraphs
  - Changing one letter in plaintext still changes one letter in ciphertext

## EARLY BLOCK CIPHER

- Playfair Cipher
  - Much harder to break because it’s encoding digraphs
  - https://learncryptography.com/classical-encryption/playfair-cipher

## EARLY ONE-WAY FUNCTIONS

- Bank transfers in the 19th century used the telegraph
- How to keep a telegraph operator from sending a false message?
- Banks developed code but this did nothing for *message integrity*
  - Money was the motivator to distinguish from *message confidentiality*
- Banks developed code books with a "test key"
  - The test key had one-way calculations for money, dates, currency, etc
  - The test key computed and the test key transmitted had to match
  - Not great by today's standards, but worked until the 1980's!!!

## ONE-WAY FUNCTIONS

- Avalanche Property – 1 bit change impacts 50% of output
- "Hard" to invert
  - Preimage resistance – cannot find input for specified output
  - Second preimage resistance – cannot find 2nd input for output
  - Collision resistance – cannot find 2 inputs with same outputs

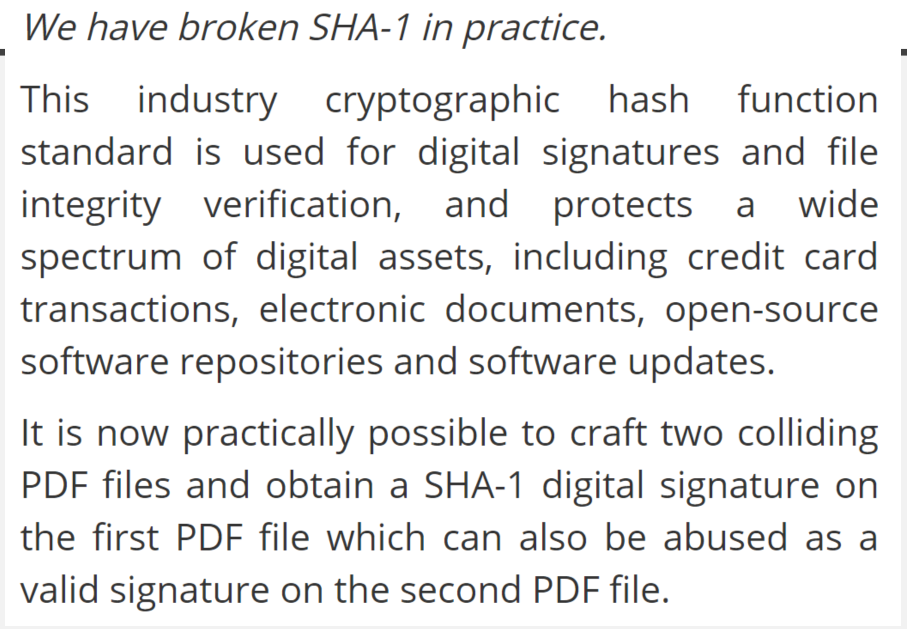
## BRUTE FORCE ON HASHES

- Iterate through possible preimages for an output:
  - Open Python3 interactive shell
  - Import hashlib
  - h = hashlib.sha1(b’a’).hexdigest()
  - bin(int(h, 16))
- Find an input *x* such that the LSb is *1*
- Find an input *x* such that the LSB’s are *01*
- Find an input *x* such that the LSB’s are *110*

## “BREAKING” HASHES

- Birthday attack on hashes
  - Find a collision in 2^(n/2) attempts
  - n is the size of the hash in bits
- Brute force ***should*** take 2^(n/2). Anything less is ***broken***
  - (This doesn’t mean that it is practical)

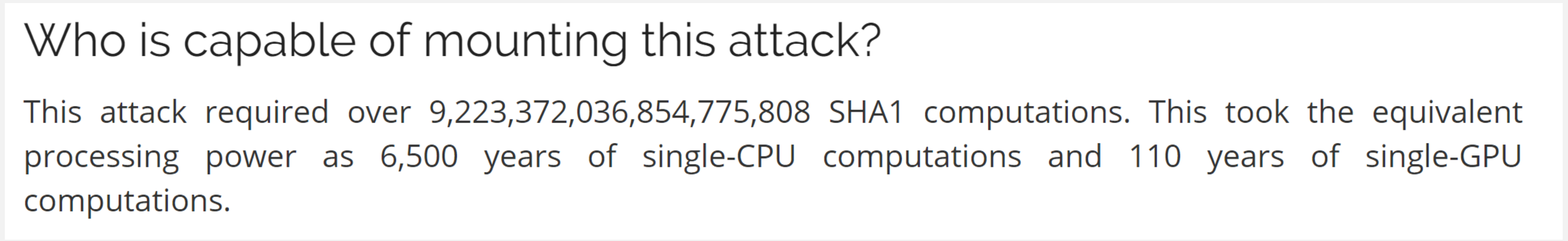
# SHA-1 IS NOW OBSOLETE



*We have broken SHA-1 in practice.*

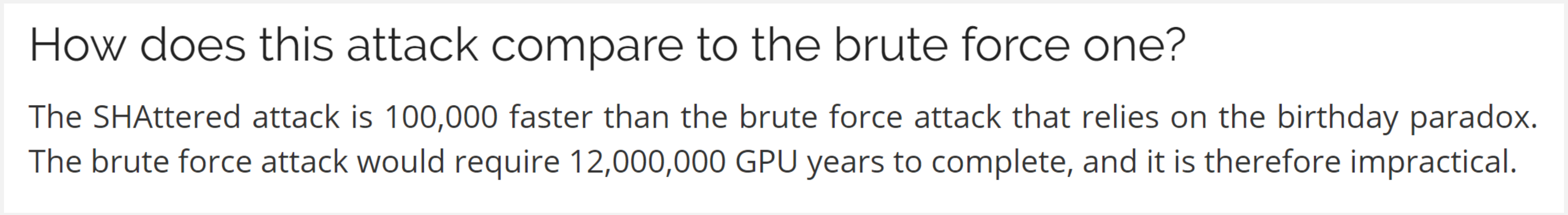
This industry cryptographic hash function standard is used for digital signatures and file integrity verification, and protects a wide spectrum of digital assets, including credit card transactions, electronic documents, open-source software repositories and software updates.

It is now practically possible to craft two colliding PDF files and obtain a SHA-1 digital signature on the first PDF file which can also be abused as a valid signature on the second PDF file.



## Who is capable of mounting this attack?

This attack required over 9,223,372,036,854,775,808 SHA1 computations. This took the equivalent processing power as 6,500 years of single-CPU computations and 110 years of single-GPU computations.



## How does this attack compare to the brute force one?

The SHAttered attack is 100,000 faster than the brute force attack that relies on the birthday paradox. The brute force attack would require 12,000,000 GPU years to complete, and it is therefore impractical.

## RESPONSIBLE REPORTING

- “We have broken sha-1 in practice”
- Irresponsible, in my opinion.
- To the average user of crypto, what does this mean?
  - Every single context/application/use?
  - Every single crypto algorithm that uses SHA1 (eg HMAC)?
- Nevertheless, ***stop using SHA-1 in all new deployments***

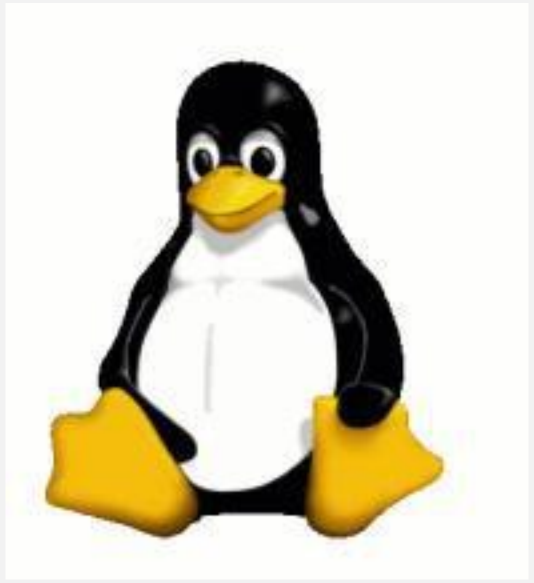
## SYMMETRIC CRYPTO

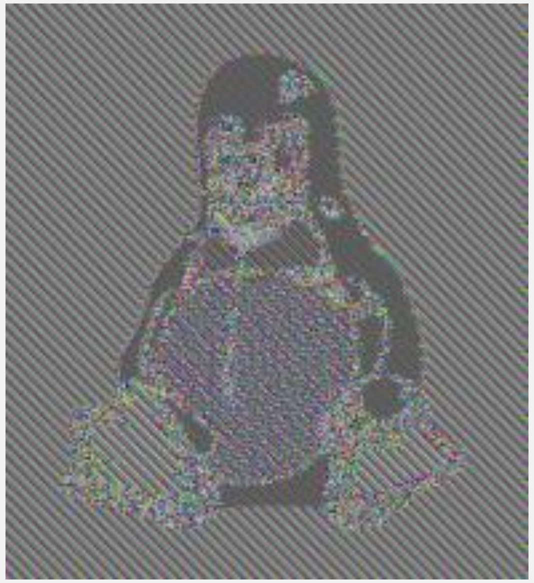
- Unlike hashing, we now assume we can ***recover*** the data
- Symmetric – Same key to encrypt and decrypt
- Parties in a communication must share a key
- Two major types:
  - Block cipher (encrypt a block at a time)
  - Stream cipher (create a stream to xor with plaintext)

## “GOOD” BLOCK CIPHERS

- Avalanche property, just like hashing
- “Large” block sizes for block ciphers
  - DES uses a 64 bit block (***deprecated***)
  - AES uses a 128 bit block
- In addition to “large” blocks, a way of “chaining” the blocks together

## AN EXAMPLE OF UNCHAINED BLOCKS



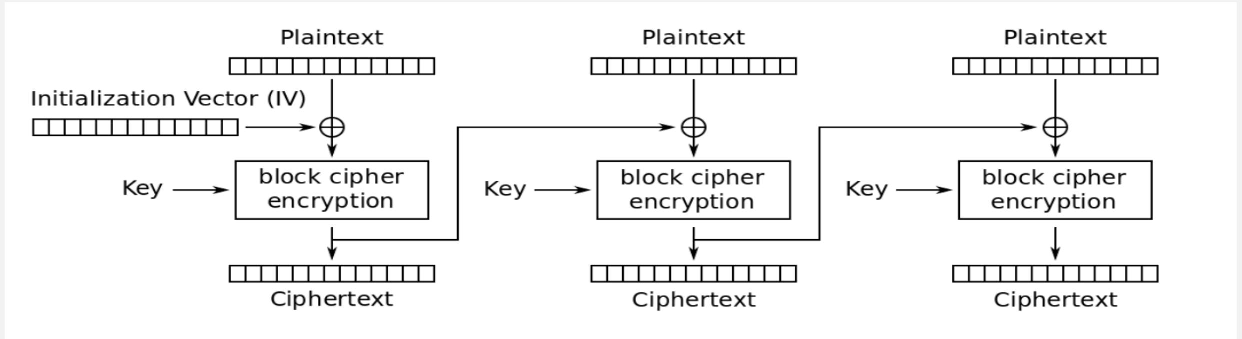


## CRYPTOGRAPHIC MODES

- This mode is called "Electronic Code Book" (ECB) mode
- *For the love of all that is holy,* ***DON'T USE IT!***
- It is simply for testing and training purposes

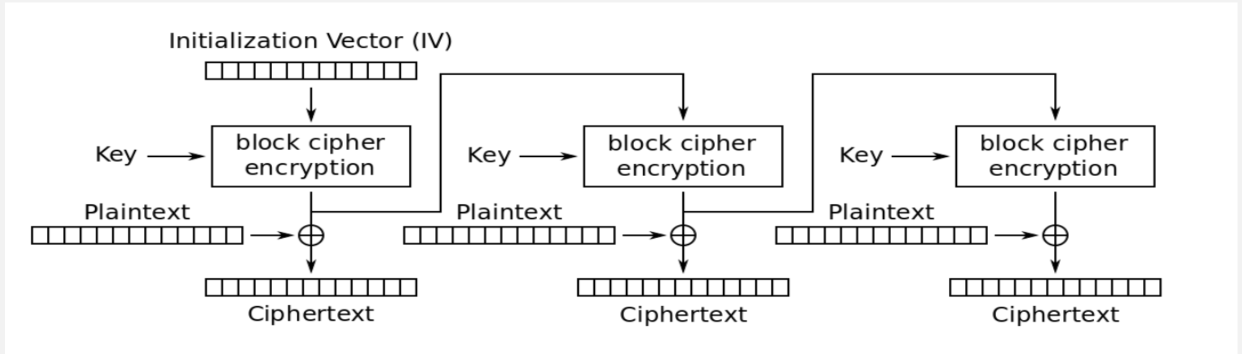
# COMMON CHAINING

- Cipher Block Chaining (CBC)



Cipher Block Chaining (CBC) mode encryption

- Output Feedback (OFB)



Output Feedback (OFB) mode encryption

## CIPHER-BLOCK-CHAINING-MODE

- Eliminates any patterns from the plaintext
- However, you can change any one cipher block and it will only affect 2 plaintext blocks
  - Don’t rely on CBC for message integrity
- The IV is critical. There was an attack on older SSL versions where the IV was predictable

## STREAM CIPHERS

- Despite being a "block" mode, OFB is a stream cipher
- One-time pad is not a streaming cipher, but similar
  - OTP is the only provably "secure" cipher (confidentiality)
  - Key must be the same length as the plaintext!
  - XOR the key with the plaintext
- Stream cipher takes a shorter key and generates key stream

# DERIVING A STREAM FROM A BLOCK CIPHER

- Output-Feedback (OFB) Mode
  - Encrypt an IV, then encrypt the output, and the output of that…
  - Xor the stream with your plaintext to get the cipher-text
    - This is called an *additive stream cipher*
- Counter (CTR) mode
  - Encrypt (IV + i) and xor (another additive stream cipher)
  - Embarrassingly parallel

## DON'T REUSE A KEY STREAM!!!

- If you reuse the same key stream, you're in big trouble.
  - C1 = K1 xor M1
  - C2 = K1 xor M2
  - C1 xor C2 = K1 xor M1 xor K1 xor M2
    - = K1 xor K1 xor M1 xor M2
    - = M1 xor M2 (if either message is natural language, easy to figure out)

## DON’T TRUST A STREAM-ENCRYPTED MESSAGE

- Suppose an attacker knows the plaintext M1
- If attacker can man-in-the-middle, can change the message
  - C1 = K xor M1
  - Attacker produces C2 = C1 xor (M1 xor M2)
    - = K xor M1 xor M1 xor M2
    - = K xor M2
- ***ALSO WORKS ON OTP (“provably secure”)***
  - Know what “secure” means (CONTEXT)

## MAC

- MAC: Message Authentication Code
- CBCMAC: last encrypted block from CBC encryption
  - Proved to be "secure" if the message length is fixed
- HMAC_k(M) = h(k xor A, h(k xor B, M))
  - A = repeated 0x36
  - B = repeated 0x5c
- Neither CBC-MAC or HMAC are *parallelizable*

## COMPOSITE MODE

- Integrity + Confidentiality
- One possibility, compute MAC with one key, and CBC with a different key
- Another possibility is CCM => Counter mode with a CBC-MAC
- AES-GCM is also pretty neat if you feel up to it
  - Parallelizable

## ASYMMETRIC PRIMITIVES

- A public/private key pair
  - Let the whole world know the public key
  - But only the "owner" knows the private key
- Unlike symmetric, what you can **DO** with asymmetric depends greatly on the algorithm
  - RSA – encryption (crypto dropbox), signatures
  - ECDSA/DSA – signatures
  - Diffie Hellman – key exchange

## RSA ENCRYPTION

- RSA allows for encrypting short messages
- Encrypted Communication between A and B
  - A and B have each other’s public key
  - A encrypts a message for B under B’s public key
  - B responds by sending A a response under A’s public key
- Works fine but…
  - It is very slow (asymmetric encryption/decryption is expensive)
- ***Used almost exclusively for <u>Key Transfer</u>*** (sending a symmetric session key)

## RSA SIGNATURES

- RSA also uses encryption for signatures
- Step 1: Publisher produces a message M
- Step 2: Publisher takes the hash of M h(M)
- Step 3: Publisher encrypts the hash with the private key $\{h(M)\}_{k^{-1}}$
- Step 4: Publisher transmits Message M and $\{h(M)\}_{k^{-1}}$ as the signature
- Step 5: A verifier decrypts the signature with Publisher's public key to obtain h(M)
- Step 6. A verifier computes their own hash of M h'(M)
- Step 7: A verifier determines the signature is valid if h'(M) = h(M)

## RSA WEAKNESSES

- ***Completely insecure when NO PADDING IS USED***
- Encryption padding schemes
  - PKCS 1.5 (***BROKEN!)***
  - OAEP
- Signature padding schemes
  - PKCS 1.5 (***BROKEN!***)
  - PSS
- Even though there are non-broken versions, RSA is being phased out
- Also, key transfer does not have “forward secrecy”

# DIFFIE HELLMAN KEY EXCHANGE

- The math version has to do with ***commutative properties.***
- Using modulo computations over *p* which is a prime with certain properties:
  - $A \rightarrow B : g^{RA} \pmod{p}$
  - $B \rightarrow A : g^{RB} \pmod{p}$
  - $A \rightarrow B : \{M\}g^{RARB}$
- A and B are the DH private keys
  - Can't be extracted from $g^{RA} \pmod{p}$
  - But, because commutative, can be combined by either side into $g^{RARB}$
- To maintain forward secrecy, a new DH key is used for ***each exchange***