Cryptology (3)

Symmetric Encryption

John York, Blue Ridge Community College

http://www.brcc.edu

Much of the information in this course came from “Understanding Cryptography” by Christoff Parr and Jan Pelzl, Springer-Verlag 2010
Much of the classical cryptography material and most Python scripts came from “Cracking Codes with Python” by Al Sweigart, NoStarch Press 2018
Also helpful, “Cryptography Engineering” by Ferguson, Schneier, and Kohno, Wiley Publishing, 2010

Symmetric Encryption Types

- Stream Cipher
  - Encrypt each bit as it is presented
  - Generate a sequence of bits of the key—key stream
  - XOR the key stream and plaintext stream (Python XOR operator is ^)
  - RC4 is stream cipher
- Block Cipher
  - Collect plaintext bits into blocks
  - Encrypt blocks, commonly 128 bits at present
  - Most current encryption uses block ciphers

Stream ciphers are not in widespread use, at least currently. Network and CPU speeds are high enough that the latency caused by waiting for 128 bits to arrive to be collected into a block is not noticeable.

Block ciphers, especially AES, are the current norm.

The XOR operation is used internally in many cryptographic algorithms and is useful when analyzing cryptography. XOR essentially adds two bits together, except it does not have a carry bit.

More terms—Initialization Vector and Nonce

- Initialization Vector (IV)
  - Unpredictable, or random number added to encryption
  - Prevents repeated encryption of plaintext from yielding same ciphertext
  - IV is generally not secret
  - Like a salt in Linux password hashes
  - IVs must never be reused—used only once
- Nonce
  - N(umber used) ONCE
  - Synonym for IV

Initialization vectors, nonces and salts (discussed later in hashes) are added for their randomness. Without them, repeated encryption of the same plaintext would result in the same ciphertext every time. An observer may not be able to decrypt the message, but they can glean information from it. If the enemy are attacked at sunrise every time they see a certain message, they can conclude the message means "attack at dawn."

An IV, nonce, or salt must be random.

Critically, an IV, nonce, or salt can only be used once. Otherwise the entire crypto system could be broken.

Standard Block Ciphers

- Data Encryption Standard (DES)
  - Block size 64 bits, key size 56 bits
  - Published 1974, broken (brute force) 1997-99, **now obsolete**
- 3DES (pronounced triple-DES) NIST calls it 3TDEA
  - DES three times
  - Useable today, provided a different key used each time
  - 3 * 56 = 168 bit key, but effective security is 112 bits
- Advanced Encryption Standard (AES)
  - Block size 128 bits, key size 128, 192, or 256 bits
  - Current NIST standard, approved 2001

DES has been broken. The algorithm is fine, but the key size is small enough (56 bits) that modern computers can use brute force attacks against it. It was broken by distributed computing in 1999, and can be broken on a decent PC today. https://en.wikipedia.org/wiki/Data_Encryption_Standard

Triple DES (3DES) is the minimum acceptable encryption approved by NIST, as of January 2016. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf It consists of running the ciphertext through the DES algorithm three times, with a different key each time. Unlike the Caesar cipher, multiple rounds of DES do improve security.
Note that the key for each of the three rounds must be different.

The effective key length for 3DES is 112 bits instead of 128 bits, due to the "meet in the middle attack." The attacker can break one round of 56 bits by attacking the left side, and the other two rounds of 56 bits (112) by attacking the right side. They keep doing this until the left and right match, which is much faster than trying to break 128 bits.

Note that in AES, you can choose different key lengths (128, 192, or 256 bits) but the block size never changes. The block size is always 128 bits or 16 bytes.

Block Cipher Goals

Key cannot be computed from ciphertext
  - Crypto term is confusion
  - Done with substitution
- A 1-bit change in plaintext changes several bits throughout entire block
  - Crypto term is diffusion
  - Done with transposition—bit permutation in DES, “Mixcolumn” in AES
- Key space is large enough to prevent brute force attacks
- Executed quickly in hardware
- Executed quickly in software

When DES is broken by brute force, the key can be computed. This breaks the first goal.

When a small change in plaintext causes a large and random change in the ciphertext, this prevents attackers from discerning part of the plaintext. For example, consider the message, “Please deposit $100”. If several messages (“Please deposit $110”, please deposit $200”, etc.) are intercepted and only the very end of the ciphertext changes, the attacker will eventually determine that the last digits are the amount.

Key space is roughly analogous to key length, provided the encryption algorithm is good and keys are randomly chosen.. It basically means, “how many times would the attacker have to guess before they break the encryption?” If the key is long, say 0x29533af76, but you only ever use two keys (0x29533af76 and 0x29533af77) the key space is two.

Rounds

- Each method (DES, AES, etc.) has an internal algorithm
- Algorithm is repeated several times, i.e. rounds
- Key is processed so that each round has a different key
- DES has 16 rounds
- AES
  - 128 bit key has 10 rounds
  - 192 bit key has 12 rounds
  - 256 bit key has 14 rounds

The number of rounds is just how many times the internal algorithm is repeated.

It is important to note that the key is manipulated so that each round has a different key. The process of determining the key for a given round is separate from the processing of the message data itself. Usually key processing is done in parallel with the message processing, and uses its own algorithm or key schedule.

AES length and rounds, https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

AES Diffusion

- Diffusion—changing a single bit of plaintext should change many bits, apparently at random, in the cipher text
- Plaintext 1 is “This is 1 secret”, hex 5468 6973 2069 7320 3120 7365 6372 6574
- Plaintext 2 is “This is 2 secret”, hex 5468 6973 2069 7320 3220 7365 6372 6574
  - (using AES ECB mode and no IV, pycryptodome in Python)

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> from Crypto.Cipher import AES
>>> import binascii
>>> key = b'This is the key!'
>>> aes_obj = AES.new(key, AES.MODE_ECB)
>>> ciphertext = aes_obj.encrypt(b'This is 1 secret')
>>> binascii.hexlify(ciphertext)
b'96a8029a5c550deae8850f76c31088b9'
>>> ciphertext = aes_obj.encrypt(b'This is 2 secret')
>>> binascii.hexlify(ciphertext)
b'0a0c517b2dfa808054145facc86219c0'
>>> |
```

This example was created using PyCryptodome, Python 3.7 on a Windows VM using Electronic Codebook (ECB) mode in AES. This code also runs in Ubuntu and CentOS. I chose Idle in Windows because I liked the colors.

The ‘b’ in front of the strings tells Python that they are to be stored as byte literals instead of strings. The aes_obj.encrypt() method will throw an error if you submit a string instead of a byte literal. Run `type 'abc'` and `type b'abc'` to see the difference.

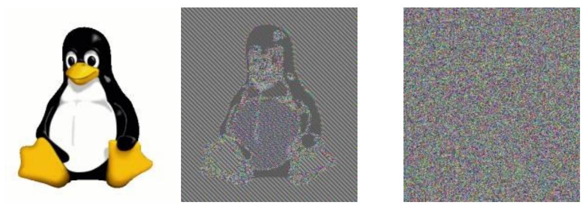
One of the biggest problems I have in coding ITSec problems is getting the data in the correct type and converting between types. The binascii module is a great help.

Normally if the encrypted data must be mailed, it is converted to base64 instead of hex characters in ASCII. In this case I chose hex (binascii.hexlify) so that it would be easy to see how much the ciphertext changed when one digit of plaintext changed.

The AES module is a ‘primitive.’ It is very picky about the data you submit. The key must be exactly 16, 24, or 32 bytes long (AES 128, AES 192, or AES 256). The data to be encrypted must be a multiple of 16 bytes (128 bits) exactly. Anything different for the key or data will cause an error.

But, it’s more complicated

- Problems when encrypting many blocks using same key, and no other changes—Electronic Codebook, or ECB mode
  - If large blocks of plaintext repeat, so does the cipher text
  - No diffusion between blocks



Left Image: plaintext

Middle image: blocks encrypted with no feedback between blocks

Right image: properly encrypted

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

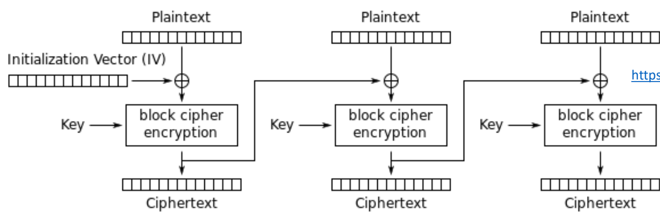
The penguin picture has many 16-byte blocks of data that are all white, all black, or all yellow. With the same key and no changes, all the white blocks have the same ciphertext, as do the all black and all yellow blocks. An attacker can use this to extract information from encrypted traffic.

General Rule

- If a block of plaintext is encrypted several times, each block of ciphertext must be different
- If a block of plaintext is always encrypted to the same ciphertext, the encryption is vulnerable
- The primary reason for having AES modes (other than ECB) is to follow this rule

Modes in Block Encryption (a few of many)

- Electronic Codebook Mode (ECB)
  - Method from previous slide—each block is encrypted with same key, independently from other blocks. Identical plaintext produces identical ciphertext
  - Bad idea
- Cipher Block Chaining (CBC)
  - Plaintext is XOR'd with the ciphertext from the previous block before encryption
  - Requires IV to XOR the first block



https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Cipher Block Chaining (CBC) mode encryption

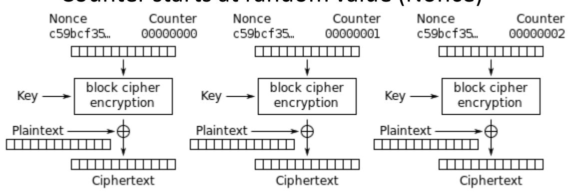
ECB is not recommended, since identical plaintext blocks produce the same ciphertext. It is nice for simple examples since you don't have to worry about initialization vectors or nonces, but not good in practice. It has no method for injecting randomness. A one block (128 bits or 16 bytes) message will always be encrypted the same way, if the key is the same.

In Cipher Block Chaining (CBC) the ciphertext from the previous block is XOR'd with the plaintext for the current block. However, the first block of plaintext does not have a previous block to XOR, so it is XOR'd with a random Initialization Vector (IV). The IV must be sent along with the ciphertext so the decryption algorithm knows where to start. The IV itself does not need to be encrypted.

The IV must be random and cannot be reused.

Modes in Block Encryption (2)

- Counter Mode (CTR)
  - Plaintext is XOR'd with a value that changes (may just increment) for each block
  - Counter starts at random value (Nonce)



Counter (CTR) mode encryption

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

- Galois Counter Mode (GCM)
  - Like CTR, "Galois" comes from multiplication over Galois Field
  - Adds a Message Authentication Code (MAC) to detect tampering

"Cryptography Engineering" by Ferguson, Schneier, and Kohno recommends CBC over CTR because IV/nonce generation is difficult. A not so random nonce or IV has a larger effect on CTR.

Counter Mode (CTR) encrypts a random nonce combined with a counter (the counter often starts at 0), and then XORs the result with the plaintext to get the ciphertext. For each subsequent block, the counter is incremented. This gives the encryption a random start and ensures that each block is encrypted differently.

The Galois Counter Mode (GCM) has an added benefit, in that computation of a Message Authentication Code (MAC) is built in. With CBC and CTR code has to be added to include a MAC. Remember the problems that WEP had because it did not include a method to detect an attacker submitting tampered data?

Note: The 'Galois' in GCM comes from mathematical operations in a Galois Extended Field. The addition and multiplication operations are redefined. If you enjoy math, you can read more in Paar, pg 90 and pg 134.

AEAD Modes in Block Encryption

- Authenticated encryption with associated data (AEAD)
- Symmetric encryption needs to protect itself from malicious input
  - Attackers can create crafted input that reveals details about the encryption
  - It may be possible for attackers to modify ciphertext to corrupt the plaintext
- Some modes create separate authentication codes that are sent along with the ciphertext. The decryption algorithm uses both the cyphertext and the code; it will generate an error if the ciphertext is corrupted.
- Some AEAD modes are GCM, CCM, EAX, and SIV
- Some modes without authentication are ~~ECB~~, CTR, and CBC

For more information on AEAD, see
https://en.wikipedia.org/wiki/Authenticated_encryption

Some of the PyCryptodome modes that include a MAC are GCM, CCM, EAX, and SIV
https://pycryptodome.readthedocs.io/en/latest/src/cipher/modern.html

Some of the pycryptodome modes that do not include a MAC are ECB, CBC, CTR
https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html

There are many more block cipher modes than the ones discussed here.

For more information on block cipher modes, see
https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation
https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html

Other Notes

- Good block ciphers include a nonlinear element
  - Otherwise if plain and cipher text are known, linear equations can find the key
- AES Competition completed in 2001
  - Winner submitted by Rijndael (pronounced rain-dahl) became AES
  - Other worthy entries—public domain, so can be used
    - Twofish
    - Serpent
    - MARS

The term nonlinear means the same thing as it does in your standard math classes. The equation y = mx + b is linear, and is easier to solve than a nonlinear equation like $y = x^2$ In a nonlinear function, $f(a + b) \neq f(a) + f(b)$.

In AES, the nonlinear component is provided by the S-boxes. $S(b_0 + b_1) \neq S(b_0) + S(b_1)$

For more information about the NIST competition for AES, see https://en.wikipedia.org/wiki/Advanced_Encryption_Standard_process

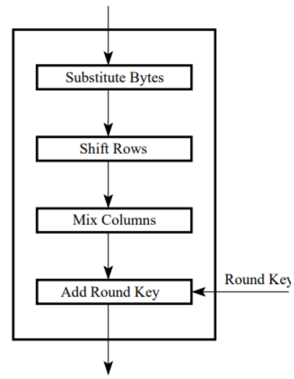
Source code for Twofish is available here. https://www.schneier.com/academic/twofish/download.html

Source code for Serpent is available here. https://www.cl.cam.ac.uk/~rja14/serpent.html

AES Extras (1)

- Data being encrypted is 4x4 matrix of bytes
- Substitute Bytes
  - Uses a lookup table (S-box) to change each byte
  - Designed to be highly non-linear
- Shift Rows
  - Row 1 unshifted, row 2 shift by 1, row 3 by 2, …
  - Adds some diffusion
- Mix Column
  - Can be thought of as matrix multiplication
  - Major diffusion element
- Add round key
  - Key is modified to be different each round
  - XOR with current key

One AES Round



https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf

The block size of AES is 16 bytes, as in $b_0, b_1, b_2, \ldots, b_{15}$. Arrange the bytes in a 4x4 matrix

$$\begin{pmatrix} b0 & b4 & b8 & b12 \\ b1 & b5 & b9 & b13 \\ b2 & b6 & b10 & b14 \\ b3 & b7 & b11 & b15 \end{pmatrix}$$

Substitute bytes uses S-boxes, described later

Shift rows. Shift each row by its row number. Row 0 doesn't move, row 1 shifts by 1, etc.

$$\begin{pmatrix} b0 & b4 & b8 & b12 \\ b5 & b9 & b13 & b1 \\ b10 & b14 & b2 & b6 \\ b15 & b3 & b7 & b11 \end{pmatrix}$$

Mix column uses matrix multiplication. However the addition and multiplication operators are Galois field operators, not standard add and multiply.

$$\begin{pmatrix} C0 \\ C1 \\ C2 \\ C3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} b0 & b4 & b8 & b12 \\ b5 & b9 & b13 & b1 \\ b10 & b14 & b2 & b6 \\ b15 & b3 & b7 & b11 \end{pmatrix}$$

AES Extras (2)

- Math behind AES is based on Extended Galois Fields
  - AES uses 2^8 elements, not a prime number
  - Modular multiplication does not always have inverse (remember Affine cipher in Cryptology 2)
  - Redefines addition, multiplication, and inverse
  - Treats bits as polynomial coefficients and uses polynomial arithmetic
- Polynomial arithmetic was used to create S-box
- A person coding AES does not do polynomial arithmetic, just uses S-box.

The complicated math (Extended Galois Fields) is used to create the S-box lookup tables, and the S-boxes are coded into the algorithm. For more information about Extended Galois Fields, see Paar pg 93.

In software implementations of AES, the S-box is constructed ahead of time, and it is incorporated into the algorithm as a lookup table.

AES Extras (3)

- Sample S-box (substitute bytes)
- 0x4a
  - Find 40 row, then 0a column
  - Result is d6
- 0xa7
  - Result is 5c
- S-box is computed ahead of time, so algorithm does not have to do computations over Galois Fields (GF)
- S-box provides substitution and is non-linear

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

https://en.wikipedia.org/wiki/Rijndael_S-box

The AES algorithm is designed around bytes, and therefore is most efficient on 8-bit computers. It is not as efficient on 32 and 64 bit computers. Often, algorithms for those computers reduce the entire round (except for Add-key) to a lookup table.

DES is based on the Feistel algorithm, DES and AES both use S-Boxes.

Now, make them into a song…