tenable®

Industrial Control Systems

Evaluating Cryptographic Implementations

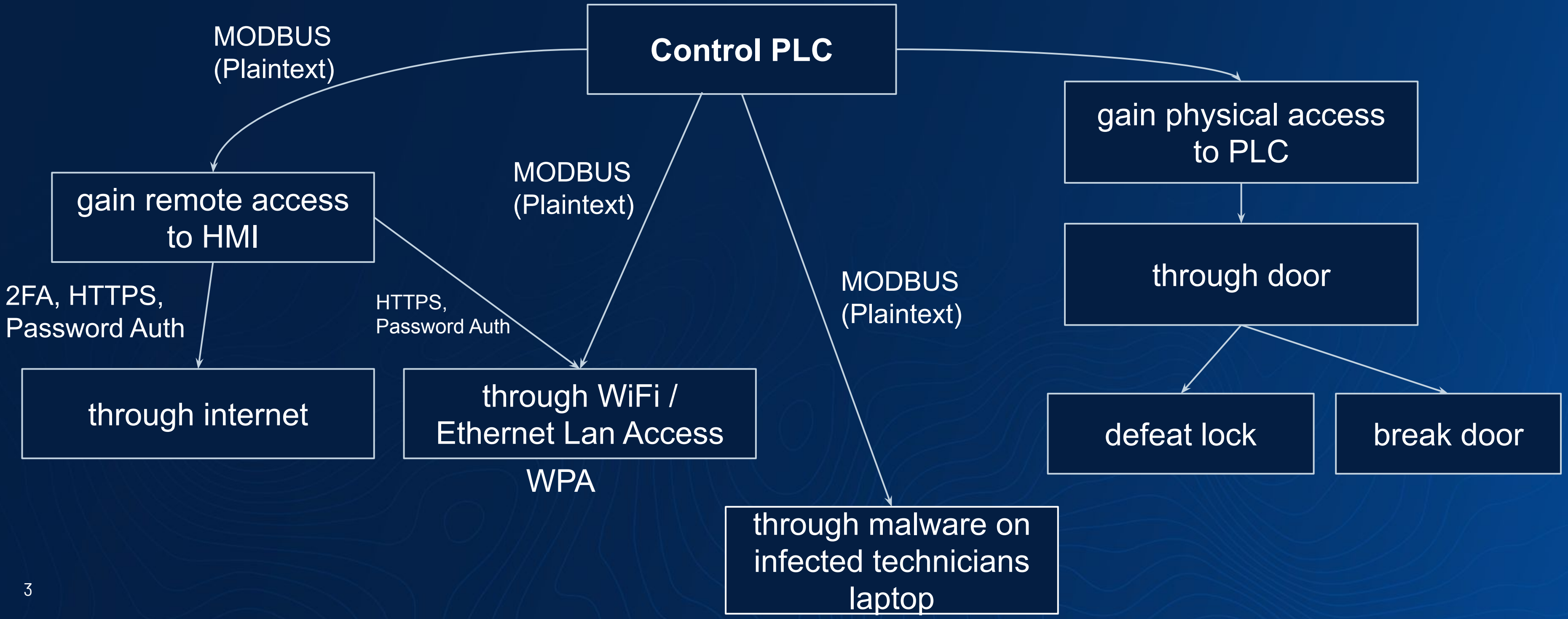
Nick Miles
nmiles@tenable.com

tenable

“A security system is only as strong as its weakest link.”
- Cryptography Engineering

- **A01:2021-Broken Access Control** moves up from the fifth position; 94% of applications were tested for some form of broken access control. The 34 Common Weakness Enumerations (CWEs) mapped to Broken Access Control had more occurrences in applications than any other category.
- **A02:2021-Cryptographic Failures** shifts up one position to #2, previously known as Sensitive Data Exposure, which was broad symptom rather than a root cause. The renewed focus here is on failures related to cryptography which often leads to sensitive data exposure or system compromise.

Example Attack Tree



Agenda

- Cryptography Basics
  - Block / Stream Ciphers
  - Hashing Algorithms
  - Digital Signatures / PKI
  - Key Exchange
  - TLS
- Case Studies
- Best Practices and Conclusions
  - Passwords and Keys
    - Secure Storage
    - 2FA

Ciphers

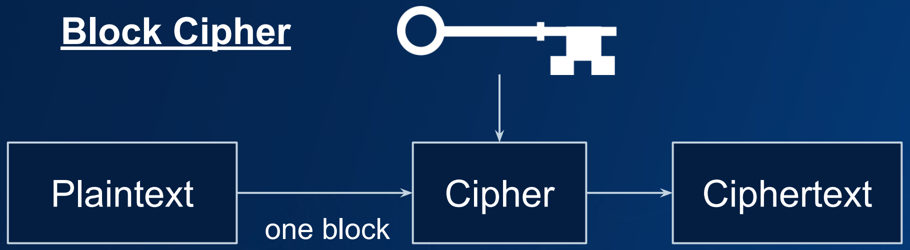
Block Ciphers

- AES (Rijndael)
  - Block Size: 128 bit
  - Key Sizes: 128, 192, 256 bits
- 3DES
  - Block Size: 64 bit
  - Key Sizes: 112 or 168 bits

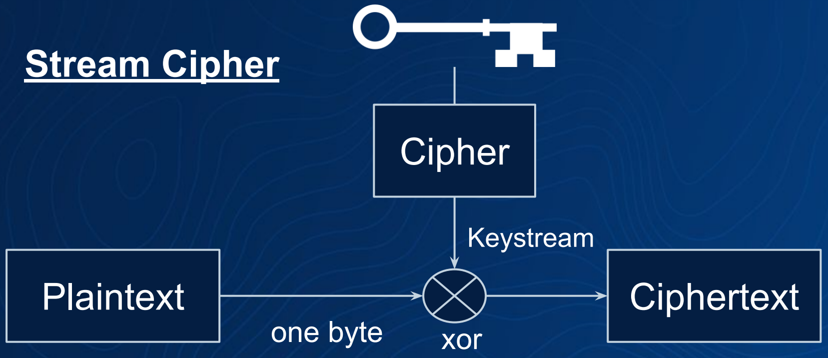
Stream Ciphers

- ChaCha20
  - State Size: 512 bit
  - Key Sizes: 128, 256 bits
- RC4
  - State Size: 2064 bits
  - Key Sizes: 40-2048 bits

Block Cipher

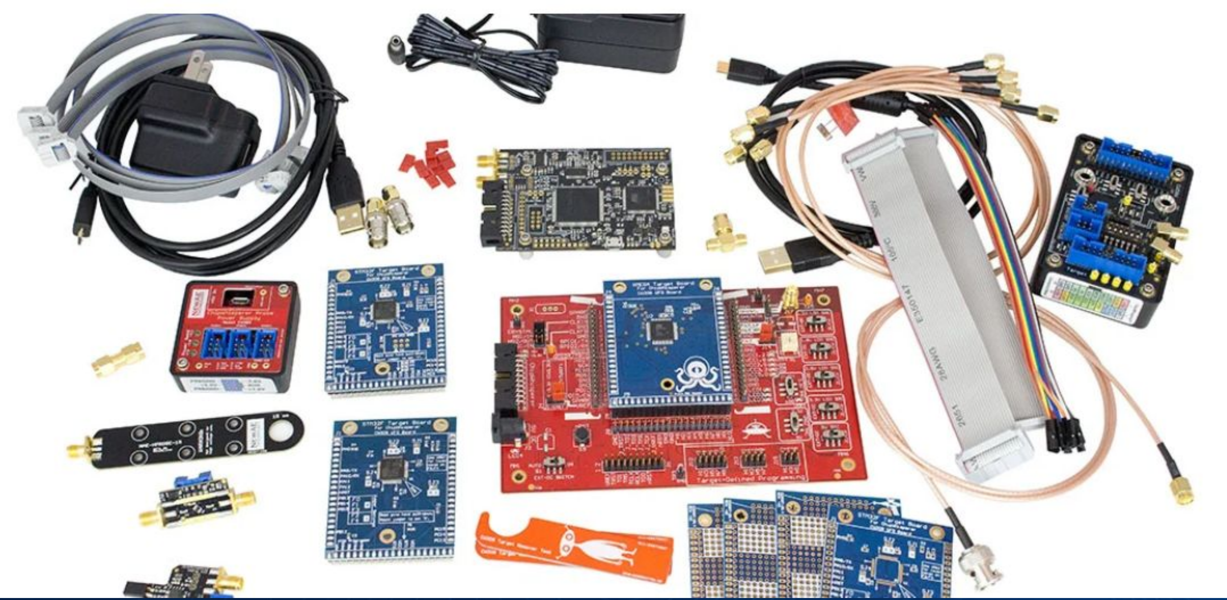


Stream Cipher



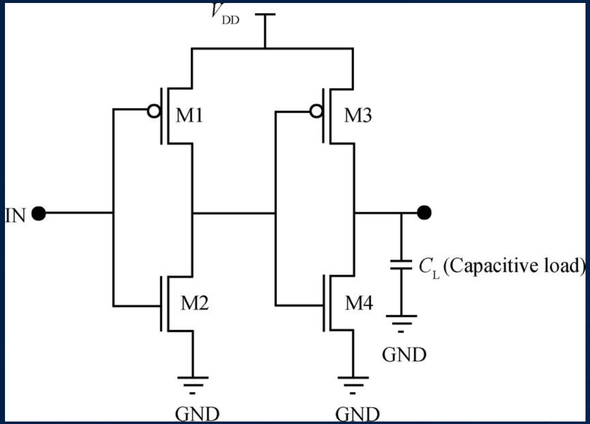
tenable

Side Channel Attacks



ChipWhisperer - https://www.newae.com/

Power Analysis Attacks



CMOS Data Bus Circuit

Driving data buses takes power.

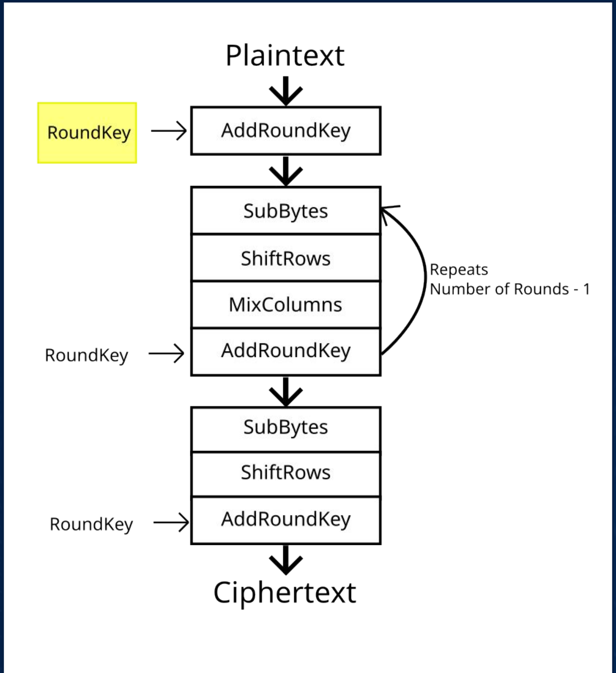
Hamming Weight Swings

11111111 -> 00000000

11111110 -> 11111111

Larger Hamming Distance = more power required

AES Block Diagram



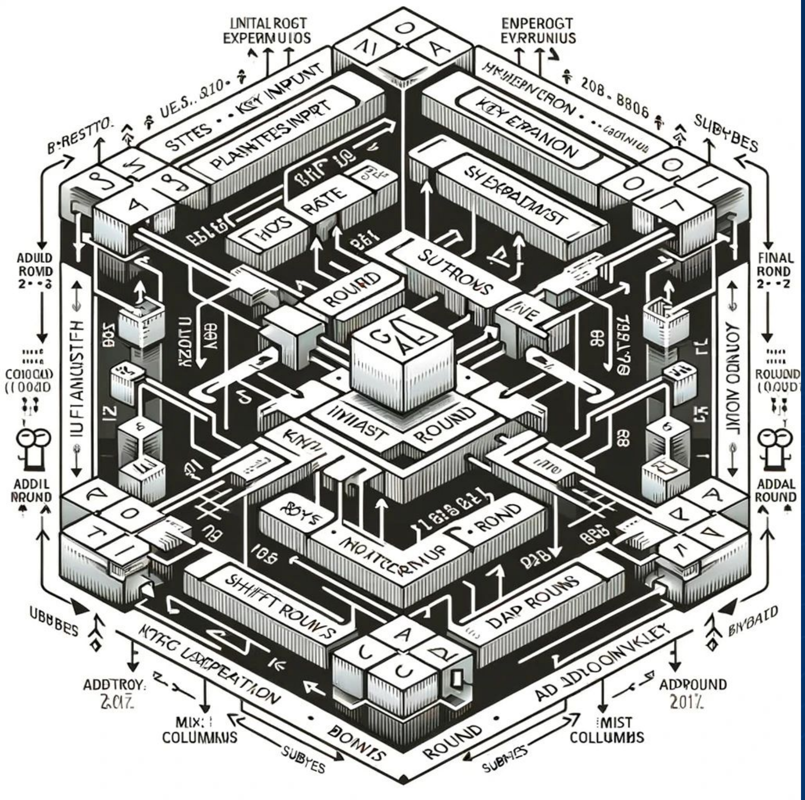
AES S-box

	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

The column is determined by the least significant nibble, and the row by the most significant nibble. For example, the value $9a_{16}$ is converted into $b8_{16}$.

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

Block Diagram AES – GPT 4



Correlation Power Analysis Attack

Input Byte	Key Guess	AddRoundKey	SubBytes	Hamming Weight	Power Trace
0xF1	0x00	0xF1	0xA1	3	
0x13	0x00	0x13	0x7D	6	
0xE2	0x00	0xE2	0x98	3	
0x83	0x00	0x83	0xEC	5	

Pearson Correlation Coefficient

Formula

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

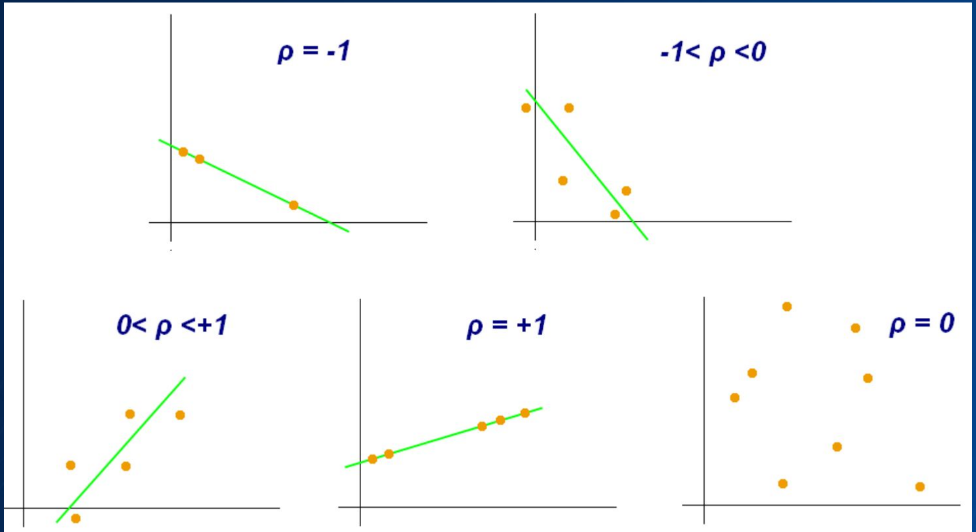
r = correlation coefficient

x_i = values of the x-variable in a sample

$\bar{x}$ = mean of the values of the x-variable

y_i = values of the y-variable in a sample

$\bar{y}$ = mean of the values of the y-variable



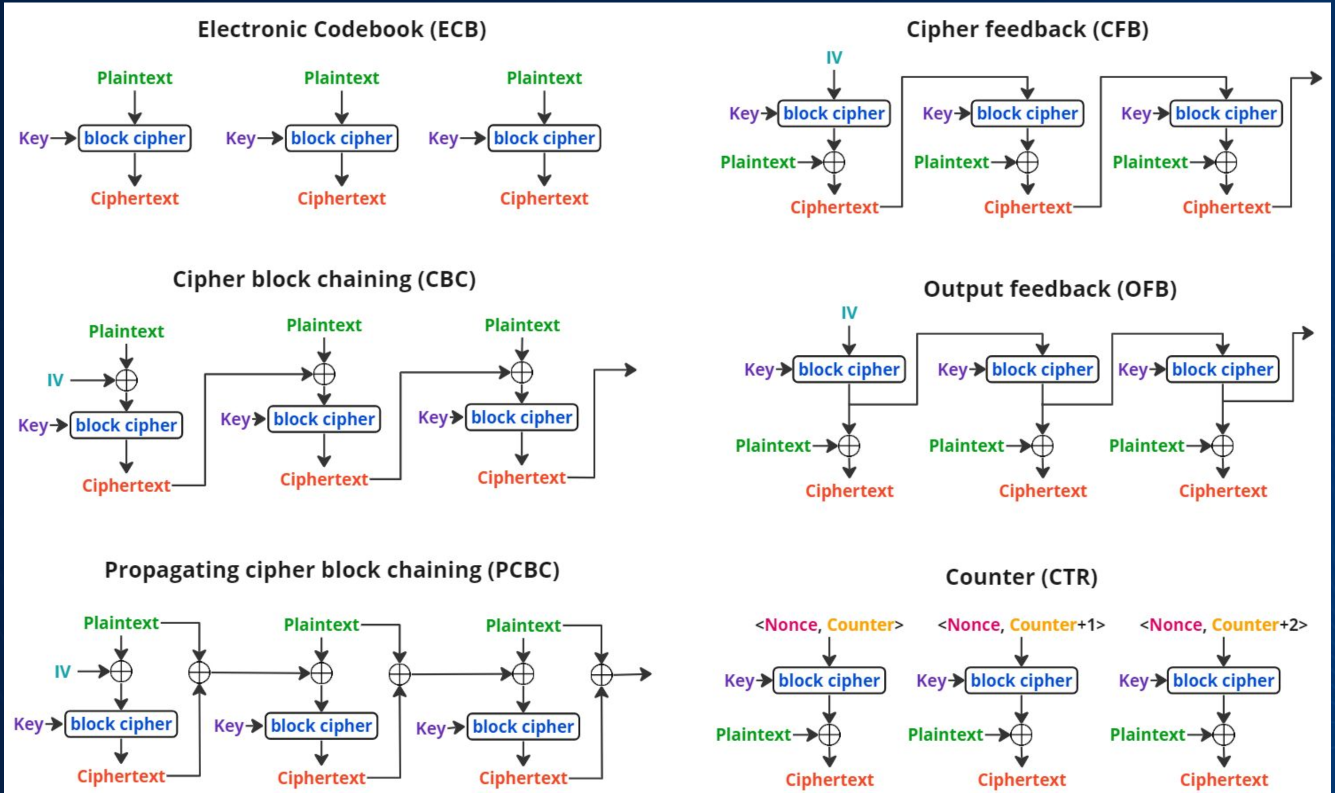
https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

tenable

Other AES issues

- Using the correct mode.
- Oracle attacks.

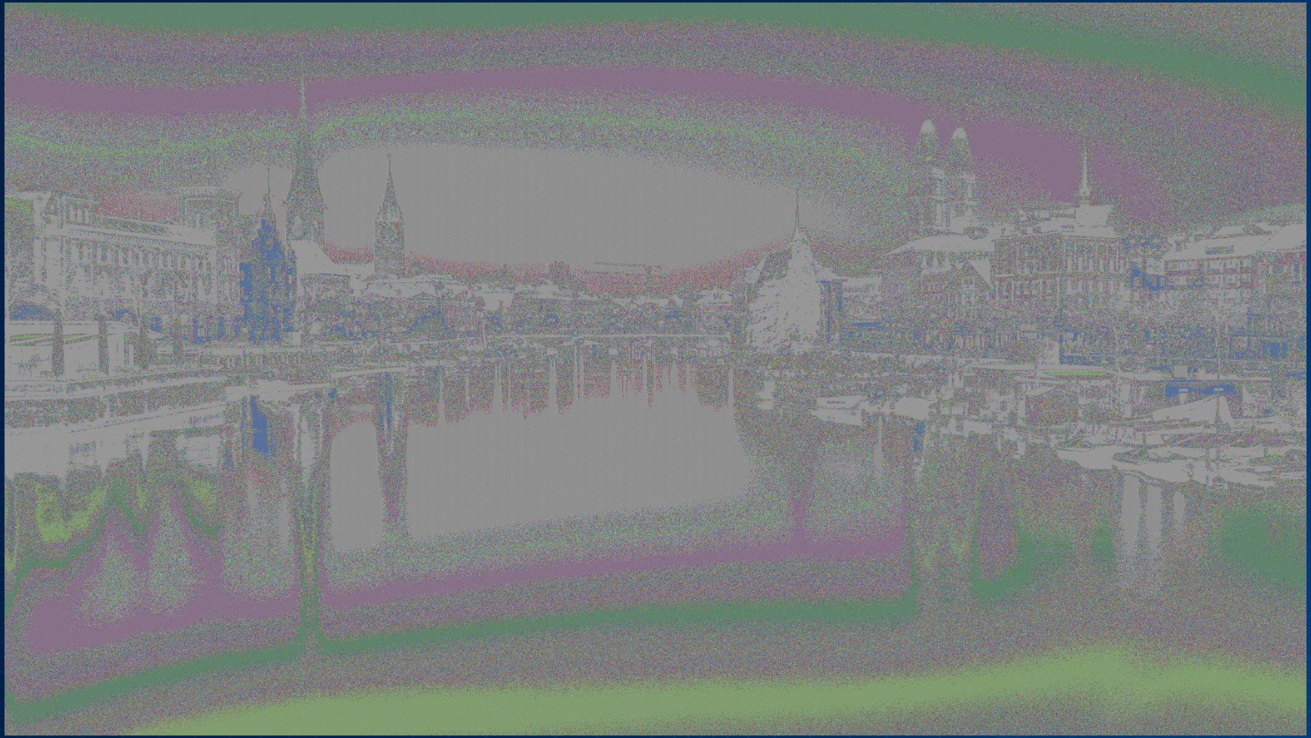
AES Modes



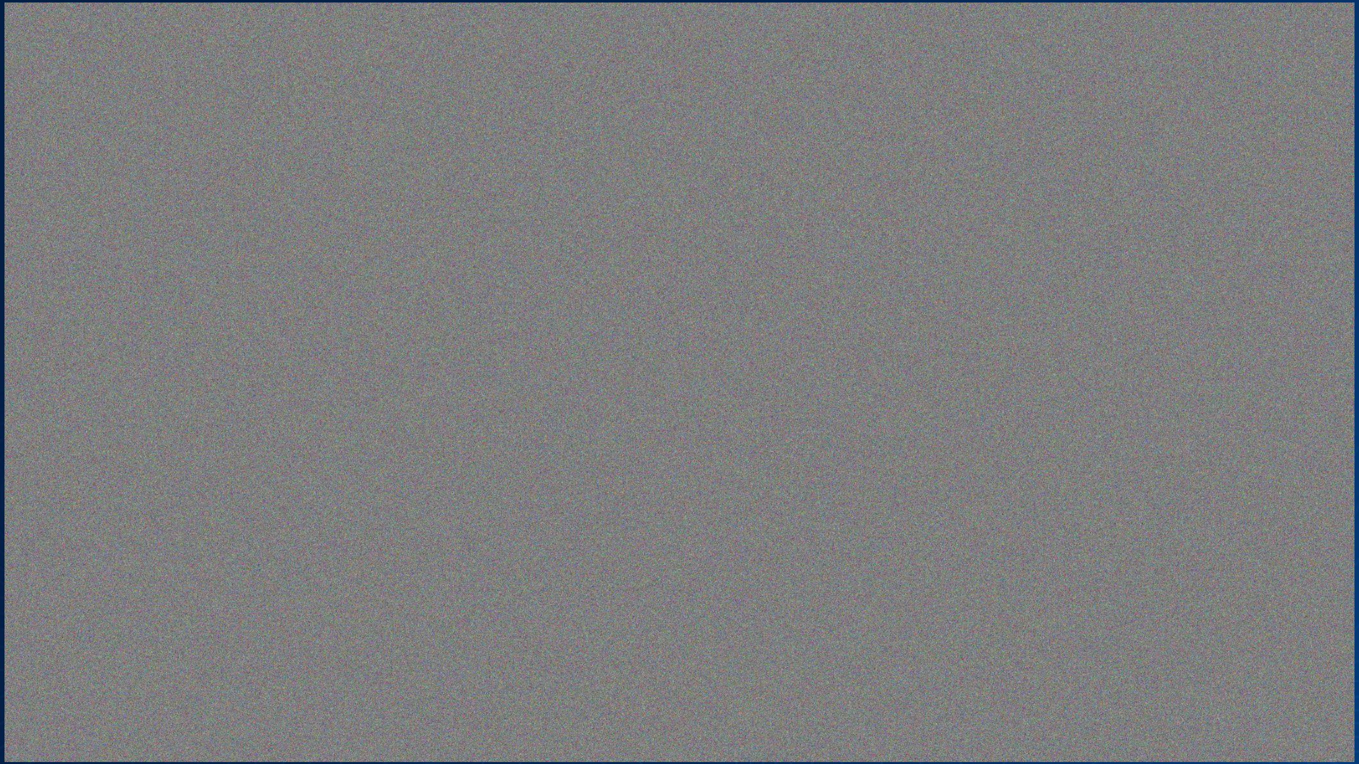
https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

AES ECB Mode - Plaintext Pixels

AES ECB Mode - Encrypted Pixel Data



AES CBC Mode - Encrypted Pixel Data



Oracle Attacks

- Oracle primitive - hotter/colder
- In practice:
  - Error Messages
  - Response Times
  - Response Length
- Types
  - Compression
  - Padding

Compression Oracle Attack

- encrypt(compress(unknown_plaintext + attacker_choosen_plaintext))
- Attacker needs to be able to view resulting encrypted traffic or traffic length.
- CRIME - SPDY, HTTPS, TLS
- BREACH - HTTP compression over HTTPS
- HTTP2 - hpack, special compression protocol mitigates these attacks
- Mitigation:
    - Don't use compression or be very selective about what is compressed

Padding Oracle Attack

PKCS#7

0x66	0x6C	0x61	0x67	0x7B	0x50	0x4B	0x43	0x53	0x37	0x5F	0x46	0x54	0x57	0x7D	**0x01**

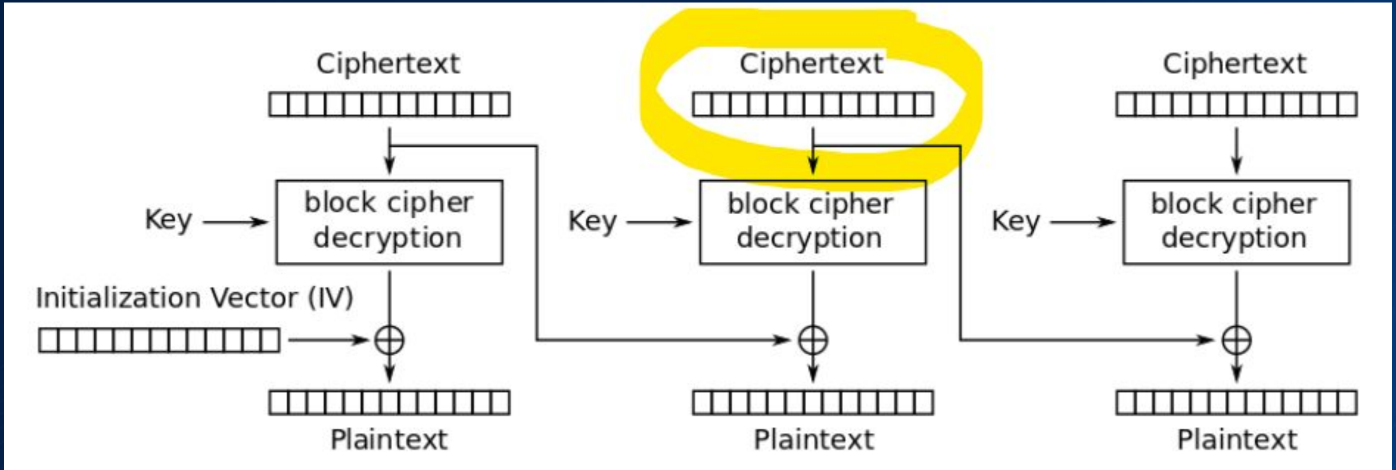
0x66	0x6C	0x61	0x67	0x7B	0x50	0x4B	0x43	0x53	0x37	0x5F	0x46	0x54	0x57	**0x02**	**0x02**

0x66	0x6C	0x61	0x67	0x7B	0x50	0x4B	0x43	0x53	0x37	0x5F	0x46	0x54	**0x03**	**0x03**	**0x03**

No padding (add block of zeros)

0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Padding Oracle



XOR Properties:

A ^ B = C

C ^ B = A

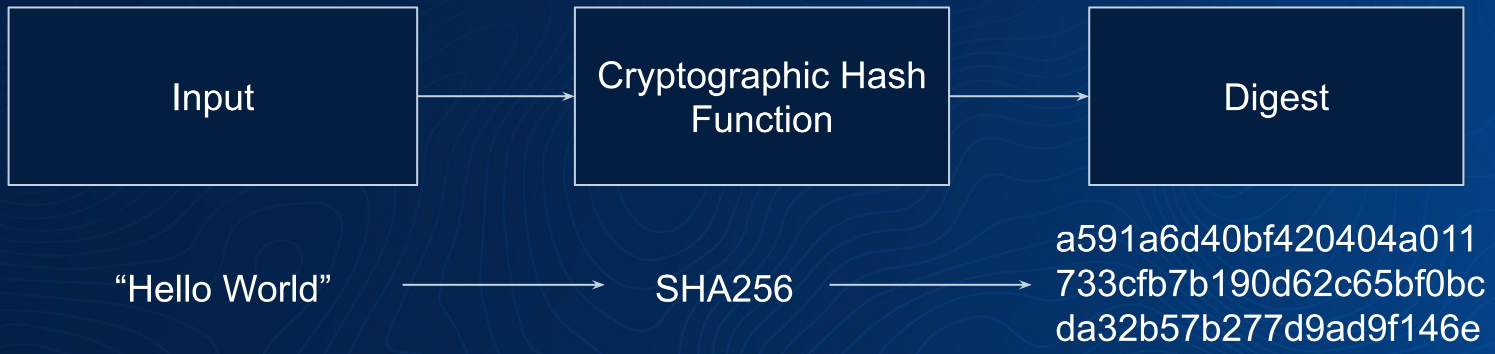
C ^ A = B

Padding Oracle – Prevention

- Don’t return an error.
- Validate message using MAC or HMAC before decryption.

Cryptographic Hash Functions

- Hashing Algorithms
  - MD5 (deprecated)
  - SHA1 (deprecated)
  - SHA 2 (256, 512), truncated 224/384
  - SHA 3



Cryptographic Hash Function Properties

- Pre-image
  - Hash functions are "one-way". If you just have a hash digest, it's difficult to a message that will hash to the same digest.

- Collision resistance
  - You should be difficult to find two messages that hash to the same digest.

Message Authentication Code (MAC)

A.

Hacky McHackface

Alice

m

Bob

m’

m

B.

Hacky McHackface

Alice

m
a = H(K, m)

Bob

m, a

m, a

tenable

MAC Attacks

Replay Attacks

Mitigations (in message):

- Nonce (random number, never repeated)
- Timestamps
- Sequence Numbers

Length Extension Attack

$m = m_1 + m_2 + \ldots + m_k$

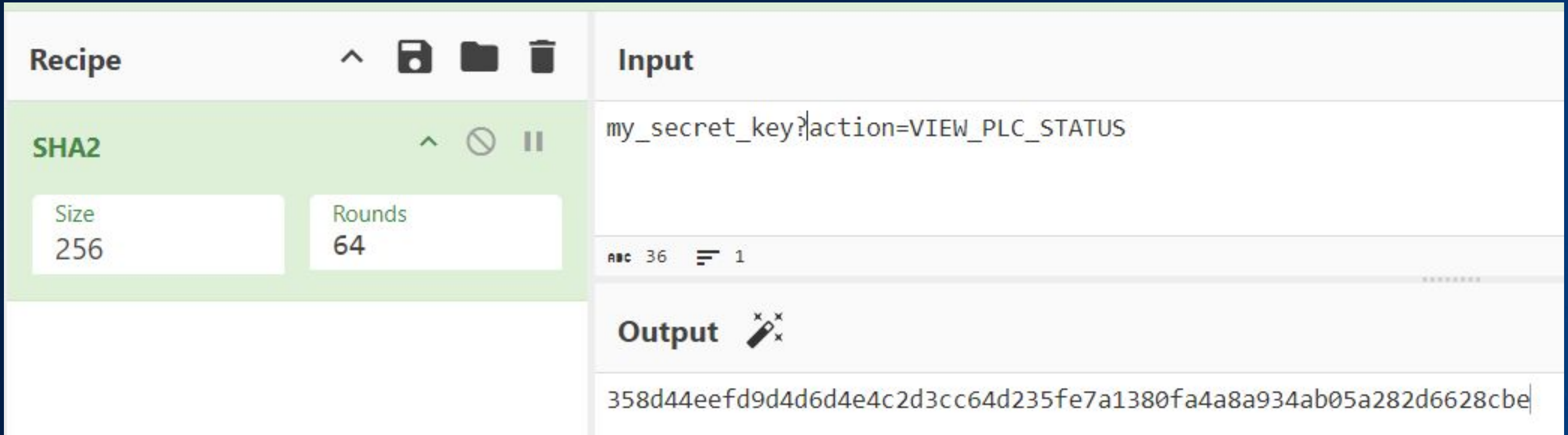
$m' = m_1 + \ldots + m_k + m_{k+1}$

$h(m') = h'(h(m), h(m_{k+1}))$

note: m’ needs to include padding and length field

tenable

Length Extension Attack - PoC



Secret: my_secret_key (13 bytes total)

Data: ?action=VIEW_PLC_STATUS

tenable

Length Extension Attack PoC Cont'd

```
import HashTools

original_data = b"?action=VIEW_PLC_STATUS"
sig = '358d44eefd9d4d6d4e4c2d3cc64d235fe7a1380fa4a8a934ab05a282d6628cbe'
append_data = b"&action=STOP_PLC"
magic = HashTools.new("sha256")
new_data, new_sig = magic.extension(
    secret_length=13, original_data=original_data,
    append_data=append_data, signature=sig
)
```

```
new_data
```

```
b'?action=VIEW_PLC_STATUS\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01 &action=STOP_PLC'
```

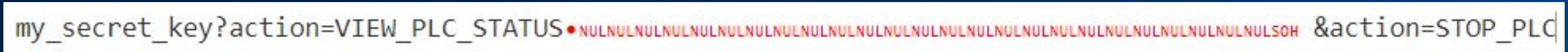
```
new_sig
```

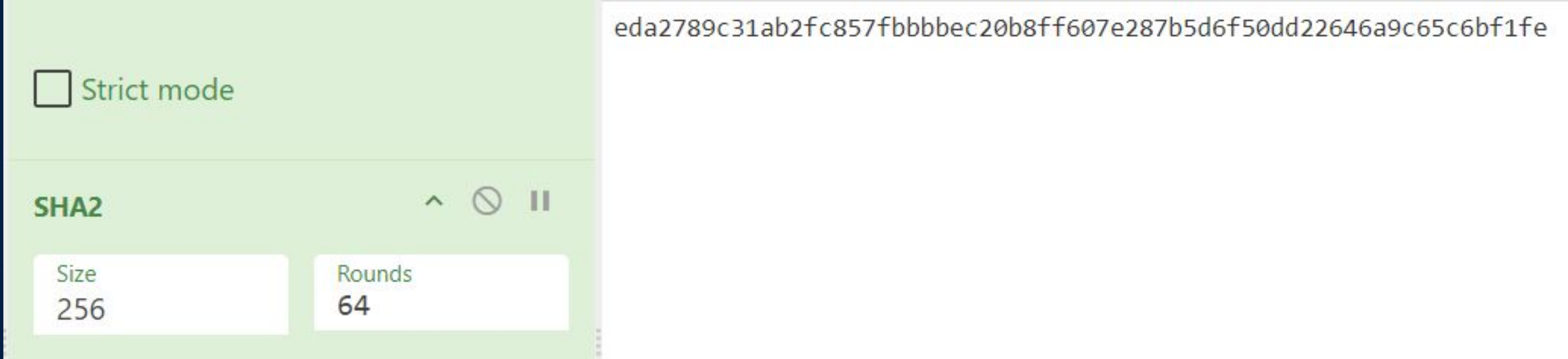
```
'eda2789c31ab2fc857fbbbbec20b8ff607e287b5d6f50dd22646a9c65c6bf1fe'
```

```
import base64
base64.b64encode(b"my_secret_key" + new_data)
```

b'bXlfc2VjcmV0X2tleT9hY3Rpb249VklFV19QTENfU1RBVFVTgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABICZhY3Rpb249U1RPUF9QTEM='

Length Extension Attack PoC (cont'd)





HMAC

- HMAC - RFC 2104

"Hash it again approach"

    - K = key, text = plaintext, H=hash function

```
We define two fixed and different strings ipad and opad as follows
(the 'i' and 'o' are mnemonics for inner and outer):

                ipad = the byte 0x36 repeated B times
               opad = the byte 0x5C repeated B times.

To compute HMAC over the data `text' we perform

                 H(K XOR opad, H(K XOR ipad, text))
```

This protects against length extension attacks, and key recovery attacks.

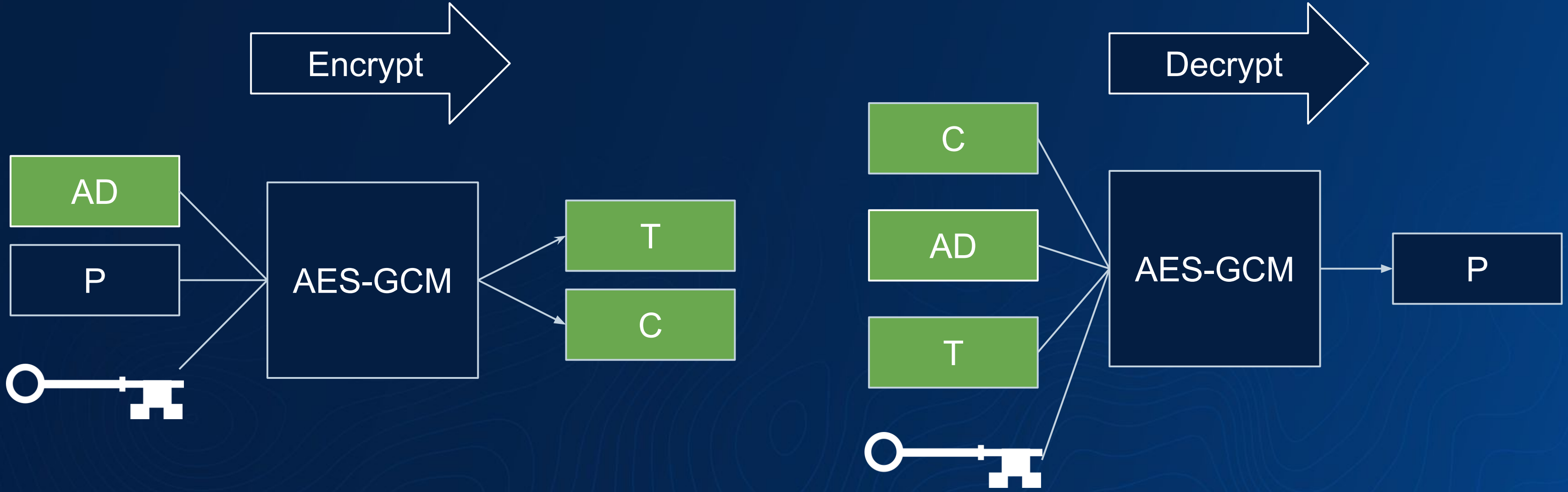
tenable

CMAC

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                                   +
+   Input    : K    ( 128-bit key )                                 +
+            : M    ( message to be authenticated )                 +
+            : len  ( length of the message in octets )             +
+   Output   : T    ( message authentication code )                 +
+                                                                   +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+   Constants: const_Zero is 0x00000000000000000000000000000000     +
+              const_Bsize is 16                                    +
+                                                                   +
+   Variables: K1, K2 for 128-bit subkeys                           +
+              M_i is the i-th block (i=1..ceil(len/const_Bsize))   +
+              M_last is the last block xor-ed with K1 or K2        +
+              n      for number of blocks to be processed          +
+              r      for number of octets of last block            +
+              flag   for denoting if last block is complete or not +
+                                                                   +
+   Step 1.  (K1,K2) := Generate_Subkey(K);                         +
+   Step 2.  n := ceil(len/const_Bsize);                            +
+   Step 3.  if n = 0                                               +
+            then                                                   +
+                 n := 1;                                           +
+                 flag := false;                                    +
+            else                                                   +
+                 if len mod const_Bsize is 0                       +
+                 then flag := true;                                +
+                 else flag := false;                               +
+                                                                   +
+   Step 4.  if flag is true                                        +
+            then M_last := M_n XOR K1;                             +
+            else M_last := padding(M_n) XOR K2;                    +
+   Step 5.  X := const_Zero;                                       +
+   Step 6.  for i := 1 to n-1 do                                   +
+                begin                                              +
+                  Y := X XOR M_i;                                  +
+                  X := AES-128(K,Y);                               +
+                end                                                +
+            Y := M_last XOR X;                                     +
+            T := AES-128(K,Y);                                     +
+   Step 7.  return T;                                              +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

RFC 4493

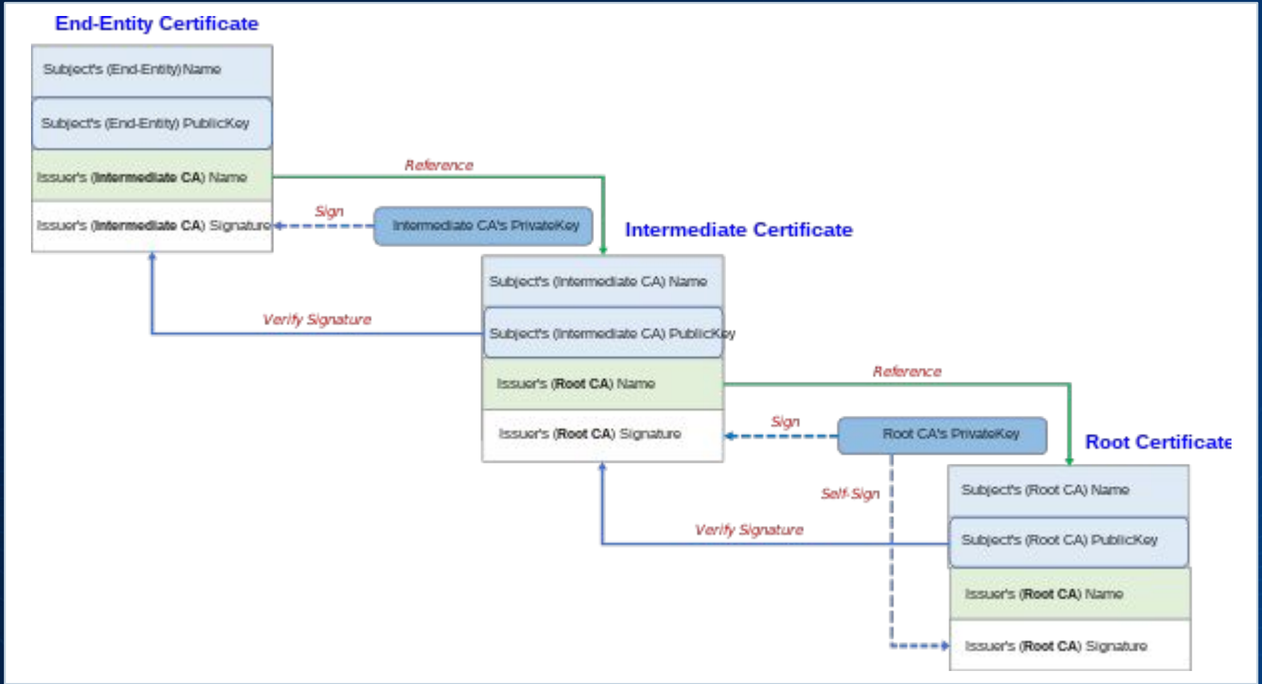
AES-GCM



- Authenticated encryption with associated data (AEAD).

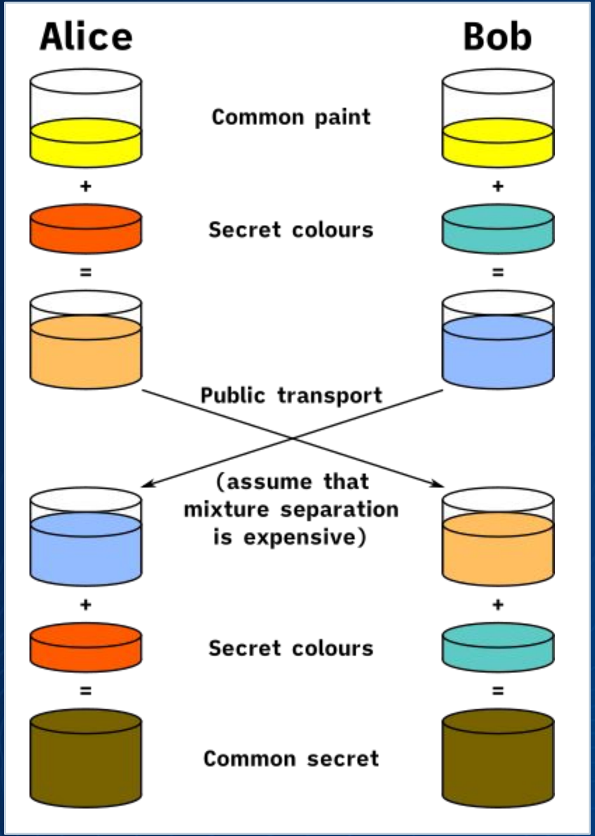
Asymmetric Algorithms

- RSA, DSA, ECDSA
  - Asymmetric Encryption Algorithms



Key Exchange

- DH (Diffie-Hellman)
- DHE or ECDHE
  - Ephemeral
- Perfect Forward Secrecy (PFS / FS)



Use TLS!!!!

- TLS 1.3 is latest version. As of April 2024, 1.1 and 1.2 are deprecated.
- If using TLS 1.3, you can be sure that it won't use any of the insecure mechanisms listed on previous slides.
- TLS 1.3 uses a shorter handshake than previous TLS versions, making it faster than previous versions.
- TLS 1.3 only uses ephemeral keys exchanged using Diffie Hellman. You can't add a key to Wireshark to decrypt this traffic, but there are other was to reverse engineer protocols using TLS 1.3.

Case Studies - Reverse Engineering Tools

Tools:

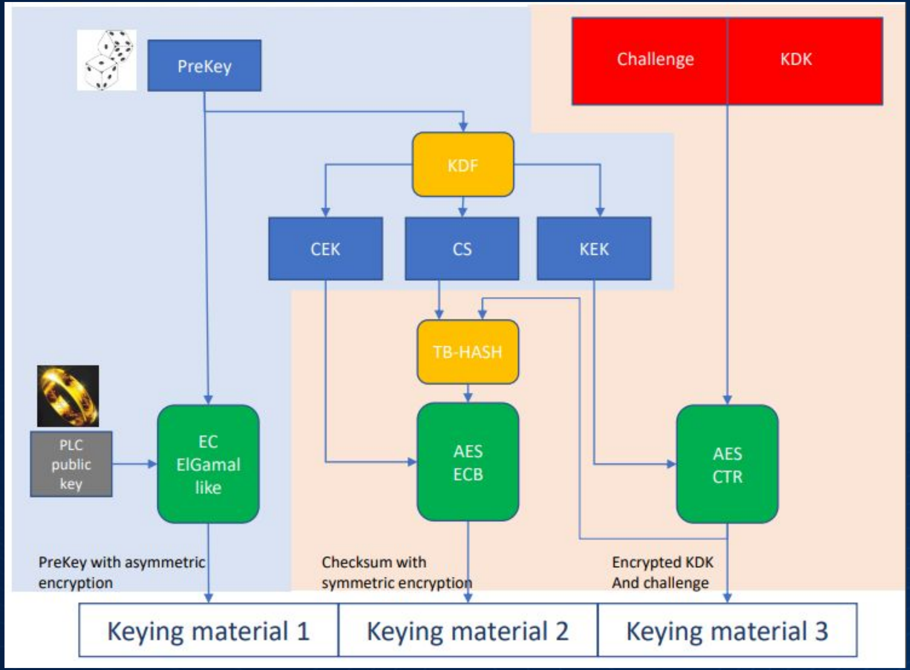
- Wireshark
- Ghidra, IDA Pro
- dnSpy (for .NET applications)
- WinDBG

Identifying TLS - s7plus

```
00000000  03 00 00 23 1e e0 00 00  00 01 00 c0 01 0a c1 02   ...#.... ........
00000010  01 00 c2 0f 53 49 4d 41  54 49 43 2d 52 4f 4f 54   ....SIMA TIC-ROOT
00000020  2d 45 53                                           -ES
    00000000  03 00 00 23 1e d0 00 01  00 01 00 c0 01 0a c1 02   ...#.... ........
    00000010  01 00 c2 0f 53 49 4d 41  54 49 43 2d 52 4f 4f 54   ....SIMA TIC-ROOT
    00000020  2d 45 53                                           -ES
00000023  03 00 00 21 02 f0 80                               ...!...
0000002A  72 01 00 12 31 00 00 05  b3 00 00 00 01 00 00 00   r...1... ........
0000003A  00 30 00 00 00 00 72 01  00 00                     .0....r. ..
    00000023  03 00 00 1f 02 f0 80 72  01 00 10 32 00 00 05 b3   .......r ...2....
    00000033  00 00 00 01 70 00 00 00  00 00 00 72 01 00 00      ....p... ...r...
00000044  03 00 04 04 02 f0 00                               .......
0000004B  16 03 01 00 ea 01 00 00  e6 03 03 53 0d cf 7e 41   ........ ...S..~A
0000005B  af c3 9c 28 b1 00 9f 15  97 05 ed 0e 99 99 4d 76   ...(.... ......Mv
0000006B  1b dc ec c3 34 39 6d 85  00 4b f9 20 94 06 8c 88   ....49m. .K. ....
0000007B  72 5a 81 cf f3 5e 7a 8e  f0 d5 a5 6b e4 95 35 80   rZ...^z. ...k..5.
0000008B  8a 54 77 25 38 3f bc a5  fe c7 b0 f3 00 26 c0 2b   .Tw%8?.. .....&.+
0000009B  c0 2f c0 2c c0 30 cc a9  cc a8 c0 09 c0 13 c0 0a   ./.,.0.. ........
000000AB  c0 14 00 9c 00 9d 00 2f  00 35 c0 12 00 0a 13 01   ......./ .5......
000000BB  13 02 13 03 01 00 00 77  00 05 00 05 01 00 00 00   .......w ........
000000CB  00 00 0a 00 0a 00 08 00  1d 00 17 00 18 00 19 00   ........ ........
```

https://blog.viettelcybersecurity.com/security-wall-of-s7commplus-part-1/

Siemens DIY Crypto



https://blog.viettelcybersecurity.com/security-wall-of-s7commplus-part-1/

tenable

Entropy Calculation

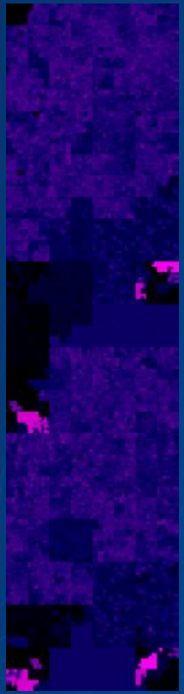
$$H(X) := -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

the average amount of information contained in message

Shannon entropy: 4.380584051175847

- 0 represents no randomness (i.e. all the bytes in the data have the same value) whereas 8, the maximum, represents a completely random string.
- Standard English text usually falls somewhere between 3.5 and 5.
- Properly encrypted or compressed data of a reasonable length should have an entropy of over 7.5.

The following results show the entropy of chunks of the input data. Chunks with particularly high entropy could suggest encrypted or compressed sections.



tenable

StartTLS

```
SReqStartTLS

03 00 00 21 02 f0 80

72 # protocol (S7 plus)
01 # Connection Packet
00 12 # Data Length
31 # Type (request)

00 00 # Null

05 b3 # starttls funccode
00 00 00 01 # sequence number

00 00 00 00 30 00 00 # ?

00 00 72 01 00 00 # frame boundary

SReqStartTLS Response

03 00 00 1f 02 f0 80

72 # S7 Plus
01 # Connection
00 10 # Data Length
32 # Type (response)

00 00 # Null

05 b3 # starttls

00 00 00 01 # sequence number

70 00 00 00 # ?

00 00 00 72 01 00 00 # frame boundary
```

```
00000023  03 00 00 21 02 f0 80                                 ...!...
0000002A  72 01 00 12 31 00 00 05  b3 00 00 00 01 00 00 00     r...1... ........
0000003A  00 30 00 00 00 00 72 01  00 00                       .0....r. ..
```

```
else if (iVar2 == 0x5b3) {
  puVar5 = (undefined8 *)(**(code **)(*plVar3 + 8))(plVar3,0x1c8);
  if (puVar5 == (undefined8 *)0x0) {
    uVar4 = 0xa00f26000121fffc;
  }
  else {
    FUN_00151e60(puVar5);
    puVar5[0x34] = 0;
    puVar5[0x35] = puVar5;
    *(undefined *)(puVar5 + 0x36) = 1;
    puVar5[0x37] = 0;
    *(undefined *)(puVar5 + 0x38) = 0;
    *puVar5 = OMS::SReqStartTLS::vftable;
    puVar5[0x33] = OMS::SReqStartTLS::vftable;
    puVar5[0x22] = puVar5 + 0x33;
    *param_3 = puVar5;
    uVar4 = 0;
  }
}
```

TLSHandler
- continue_execution_cb
- decrypt
- deliver_verification_info
- do_handshake
- finish_longrunner_cb
- handle_result
- postprocess_read
- postprocess_write
- preprocess_read
- preprocess_write
- RTTI_Base_Class_Array
- RTTI_Base_Class_Descriptor_at_(0,-1,0,64)
- RTTI_Class_Hierarchy_Descriptor
- RTTI_Complete_Object_Locator
- RTTI_Complete_Object_Locator
- RTTI_Type_Descriptor
- s_ssl_error
- starttls
- tls_wakeup
- vftable
- vftable
- vftable_meta_ptr
- vftable_meta_ptr

tenable

TLS Handshake (s7plus opportunistic / starttls TLS example)

```
0000004B  03 00 00 ce 02 f0 80 16  03 01 00 c2 01 00 00 be   ........ ........
0000005B  03 03 9a 17 37 40 b4 b8  7b 69 93 0e 98 ab 06 b0   ....7@.. {i......
```

Client Hello

```
00000042  03 00 03 c1 02 f0 80 16  03 03 00 7a 02 00 00 76   ........ ...z...v
00000052  03 03 78 7e fd b1 3d d2  ab 3e 3b 5d 23 69 ea 26   ..x~..=. .>;]#i.&
```

Server Hello

```
00000120  03 00 00 47 02 f0 80 14  03 03 00 01 01 17 03 03   ...G.... ........
00000130  00 35 fa b6 a8 9a ad 3e  58 9d 9f c0 55 e4 52 53   .5.....> X...U.RS
```

Change Cipher Spec - 0x14 (for backwards compatibility)

```
00000167  03 00 00 fa 02 f0 80 17  03 03 00 ee 05 7d b6 d6   ........ .....}..
00000177  83 d6 ab b3 aa 71 89 f1  b1 67 5b 31 08 37 01 0a   .....q.. .g[1.7..
```

0x17 - TLS Application Data

```
00000403  03 00 01 c5 02 f0 80 17  03 03 00 da 00 c1 fb 12   ........ ........
00000413  08 39 cf 67 41 de b4 14  e7 8a 10 5f 75 c7 51 92   .9.gA... ..._u.Q.
```

TLS Headers

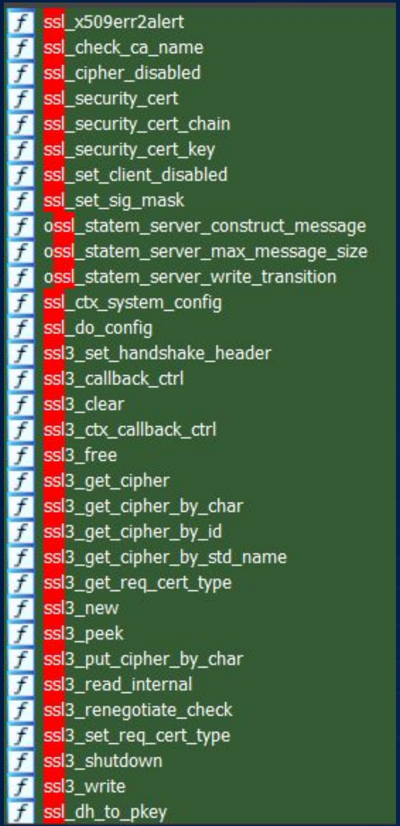
```
   struct {
       ContentType type;
       ProtocolVersion legacy_record_version;
       uint16 length;
       opaque fragment[TLSPlaintext.length];
   } TLSPlaintext;

type:  The higher-level protocol used to process the enclosed
   fragment.

legacy_record_version:  MUST be set to 0x0303 for all records
   generated by a TLS 1.3 implementation other than an initial
   ClientHello (i.e., one not generated after a HelloRetryRequest),
   where it MAY also be 0x0301 for compatibility purposes.  This
   field is deprecated and MUST be ignored for all purposes.
   Previous versions of TLS would use other values in this field
   under some circumstances.

length:  The length (in bytes) of the following
   TLSPlaintext.fragment.  The length MUST NOT exceed 2^14 bytes.  An
   endpoint that receives a record that exceeds this length MUST
   terminate the connection with a "record_overflow" alert.
```

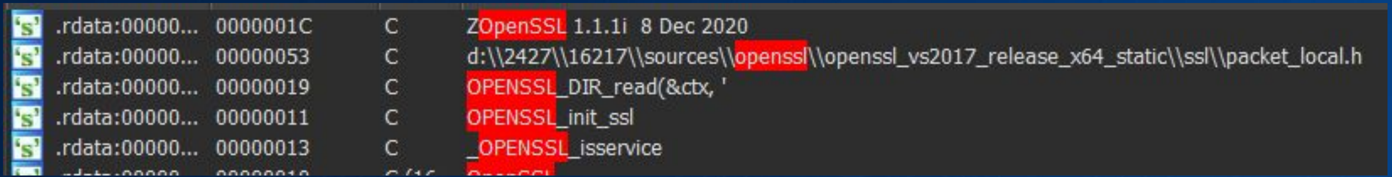
```
16 03 01 00 ea
```

Type: 0x16
Version: 1.3
Length: 0x00ea (234)

tenable

s7plus TLS - v17 and up





tenable

Reverse Engineering Protocols With TLS 1.3 - s7plus

```
$$ buf is rdx, num is r8d  [breakpoints in TLSHandler::decrypt() after SSL_read() call]
bu OMSp_core_managed+0x000e11e4 ".echo \"decrypted plaintext buf\"; .frame; db rdx L 512; .echo
\"decrypted plaintext num\"; r eax; gc"

$$ buf is rdx, num is r8d  [in TLSHandler::preprocess_write(), breakpoints around SSL_write() call]
bu OMSp_core_managed+0x000dfb55 ".echo \"encrypt plaintext buf\"; .frame; db rdx L 512; gc"
bu OMSp_core_managed+0x000dfb5d ".echo \"encrypt plaintext num\"; .frame; r eax; gc"
```

Reverse Engineering S7Plus

```c
undefined8 FUN_005128e0(undefined4 *param_1)

{
  undefined4 uVar1;
  longlong lVar2;
  int iVar3;

  if (*(longlong *)(param_1 + 2) == 0) {
    ERR_put_error(0x14,0xa4,0xbc,"ssl\\ssl_lib.c",580);
    return 0;
  }
  iVar3 = FUN_0051e120();
  if (iVar3 != 0) {
    FUN_0051d9a0(*(undefined8 *)(param_1 + 0x142));
    *(undefined8 *)(param_1 + 0x142) = 0;
  }
  FUN_0051d9a0(*(undefined8 *)(param_1 + 0x144));
  *(undefined8 *)(param_1 + 0x144) = 0;
  CRYPTO_free(*(void **)(param_1 + 0x146),"ssl\\ssl_lib.c",590);
  *(undefined8 *)(param_1 + 0x146) = 0;
  *(undefined8 *)(param_1 + 0x148) = 0;
  param_1[0x136] = 0;
  *(undefined8 *)(param_1 + 0x5d2) = 0;
  param_1[0x15c] = 0;
  param_1[0x32] = 0;
```

```c
int ossl_ssl_connection_reset(SSL *s)
{
    SSL_CONNECTION *sc = SSL_CONNECTION_FROM_SSL(s);

    if (sc == NULL)
        return 0;

    if (ssl_clear_bad_session(sc)) {
        SSL_SESSION_free(sc->session);
        sc->session = NULL;
    }
    SSL_SESSION_free(sc->psksession);
    sc->psksession = NULL;
    OPENSSL_free(sc->psksession_id);
    sc->psksession_id = NULL;
```

WinDBG Output

```
decrypted plaintext buf
00 0000002b`15eff920 00007ffb`e1230f62     OMSp_core_managed+0xe11e4
000001cd`00f53458  72 02 00 15 32 00 00 05-86 00 00 00 04 34 00 00  r...2.......4..
000001cd`00f53468  00 04 a0 00 05 00 00 00-00 72 02 00 00 17 46 06  .........r....F.
000001cd`00f53478  bd 7f b4 24 9b af 6a 11-02 11 a9 c8 24 b2 31 42  ...$..j.....$.1B
000001cd`00f53488  37 42 30 38 34 37 44 31-31 36 39 34 a3 82 2b 00  7B0847D11694..+.
000001cd`00f53498  04 01 a3 82 2d 00 15 1c-4f 4d 53 50 5f 31 32 2e  ....-...OMSP_12.
000001cd`00f534a8  30 30 2e 30 31 2e 30 37-5f 33 35 2e 30 37 2e 30  00.01.07_35.07.0
000001cd`00f534b8  30 2e 30 31 a3 82 2f 10-02 14 9f fc e1 9b 28 53  0.01../.......(S
...
...
000001cd`00f53958  b0 4f 75 e1 c9 7f 01 80-f0 00 00 00 c9 01 00 00  .Ou.............
000001cd`00f53968  41 4e                                            AN
decrypted plaintext num
eax=1d
encrypt plaintext buf
00 0000002b`15eff820 00007ffb`e129148c     OMSp_core_managed+0xdfb55
000001cd`00ef6470  72 02 00 36 31 00 00 04-f2 00 00 00 05 70 00 0c  r..61........p..
000001cd`00ef6480  86 34 70 00 0c 86 01 8e-6f 00 04 a0 00 00 00 04  .4p.....o.......
000001cd`00ef6490  e8 89 69 00 12 00 00 00-00 89 6a 00 13 00 89 6b  ..i.......j....k
000001cd`00ef64a0  00 04 00 00 00 04 00 00-00 00 72 02 00 00 40 82  ..........r...@.
000001cd`00ef64b0  3f 00 15 00 82 40 00 15-1a 31 3b 36 45 53 37 20  ?....@...1;6ES7
000001cd`00ef64c0  35 31 31 2d 31 41 4b 30-32 2d 30 41 42 30 3b 56  511-1AK02-0AB0;V
000001cd`00ef64d0  32 2e 39 82 41 00 03 00-03 00 02 00 04 01 00 00  2.9.A...........
000001cd`00ef64e0  00 04 e8 89 69 00 12 00-00 00 00 89 6a 00 13 00  ....i.......j...
000001cd`00ef64f0  89 6b 00 04 00 00 00 00-00 00 72 02 00 00 50 50  .k........r...PP
000001cd`00ef6500  38 5f 31 31 38 37 31 37-39 33 35 39 a3 82 2b 00  8_1187179359..+.
...
000001cd`00ef6970  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000001cd`00ef6980  00 00                                            ..
encrypt plaintext num
00 0000002b`15eff820 00007ffb`e129148c     OMSp_core_managed+0xdfb5d
eax=3e
```

Beware Deprecated Algorithms

```
public j(string key)
{
this.a = new TripleDESCryptoServiceProvider();
this.a.Key = this.a(key, this.a.KeySize / 8);
this.a.IV = this.a("", this.a.BlockSize / 8);
}
```

```
public [redacted]_Encryption()
{
    this.basekey = "[redacted]";
}
```

- NIST has deprecated **DES** and **3DES** for all applications.
  - AES is a a good replacement
- SHA1 and MD5 are deprecated.
  - Recommend SHA256/512 as replacement.
- RSA < 2048 bits.

tenable

Password Storage (Client Side)

```
public static String RotInput(String paramString) {
  StringBuffer stringBuffer = new StringBuffer(paramString);
  for (byte b = 0; b < stringBuffer.length(); b++)
    stringBuffer.setCharAt(b, rot13(stringBuffer.charAt(b)));
  return stringBuffer.toString();
  return paramChar;
}
private void writeUserData(PrintWriter paramPrintWriter) throws IOException {
  paramPrintWriter.println(this.HTTPUsername);
  paramPrintWriter.println(this.HTTPPassword);
}
private void writePWData(PrintWriter paramPrintWriter) throws IOException {
  paramPrintWriter.print(this.Password);
}
private void writeConfigData(PrintWriter paramPrintWriter) throws IOException {
  paramPrintWriter.print(RotInput(this.configPassword));
}
```

Password Storage (Server Side)

- Don’t store passwords, store their salted and hashed digests (using a cryptographically sounds RNG source, and FIPS compliant hash algorithm).
    - e.g. rnd_str + ‘$’ + SHA256(rnd_str+password)

- Better yet, use an algorithm designed for storing passwords that is FIPS compliant.
    - Argon2id
    - scrypt (a version of this called "yescrypt" is used in Ubuntu, see example below)
    - bycrypt

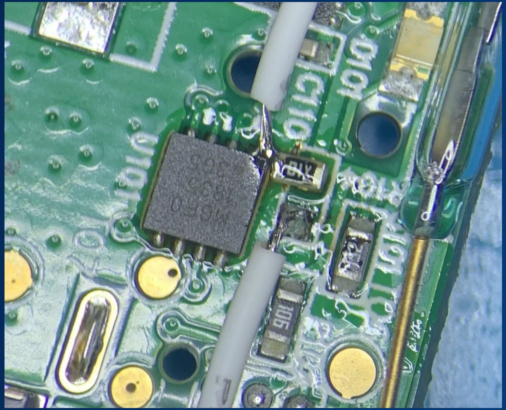
https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

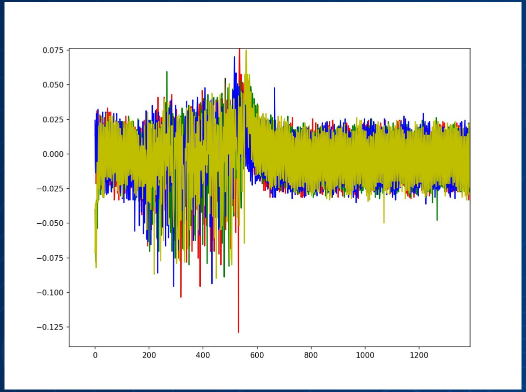
```
genius:$y$j9T$.JWhKaIhAm.ZBDPhwYRx2.$QfczucaFDPcirfeNrNNkuKjcDK3wL68ybv/juqJtwF1:19850:0:99999:7:::
```

tenable

Hardcoded Keys

```
***********************************************************************
***********************************************************************
Packet Recieved:
1602007289cb510c0222fee5ee97e51de7400d53b39603ab1b
=== PACKET DISSECTION ===
Packet          : 1602007289cb510c0222fee5ee97e51de7400d53b39603ab1b
Length          : 16 (22)
Serial          :     007289cb
Counter         :             510c02
CMAC:           :                   22fee5ee
Encrypted Data  :                           97e51de7400d53b39603
Chksum          :                                               ab1b

Data Decryption:
  AES Call 1 Res  : 92b623e7408853b3d3030171c43cd0d1
  Decrypted Data  : 05533e00008500004500

CMAC Verify:
  AES Call 2 Res  : 34d21f79bc403b8ccbaa7fb1585be8e2
  LSFR Res1: 4b6339d4938656ed442eeabcd1d7e15f
  CHK Data1  : 97e51de7400d53b39603
  CHK Data2  : dc862433d38b055ed22deabcd1d7e15f
  LSFR Res2: 04fd38836fee6b1907a090aac0e346e0
  LSFR Res2_xor: 04fd38836fee6b9907a090aac0e346b0
  LSFR Res3: 8cebfd6e76cdbee1f74ed4215c9b2e41
  AES Call 3 Res  : ae151880f76a3c9690548c62b10fad2b
  CHK Data3 (CALCULATED CMAC): 22fee5ee
     CMAC Match!

Checksum Verify:
test: 0xab1b
  Calculated Checksum:ab1b
     Checksum Verified!
***********************************************************************
***********************************************************************
```





tenable

Improper Password Authentication

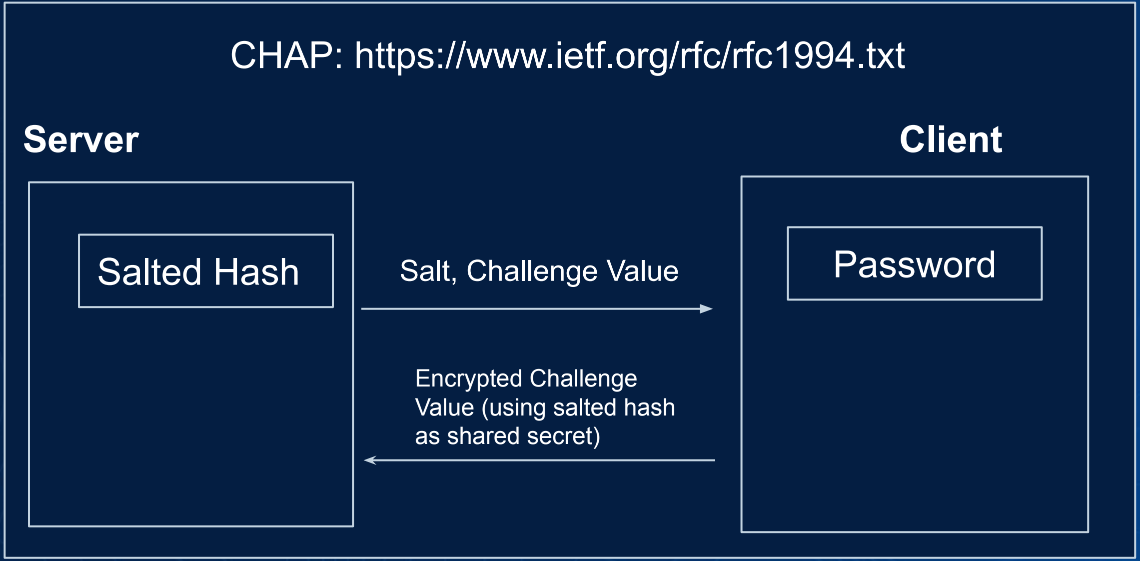
```
# Reading memory block from controller
0000042D  45 00 00 00 00 0d 00 5a  00 20 01 <redacted> 00 1a 01 00   E......Z .
......
0000043D  00 3d 00                                              .=.
    0000059D  45 00 00 00 00 44 00 5a  00 fe 01 3d 00 42 5a 74   E....D.Z ...=.BZt
    000005AD  66 64 69 41 58 67 52 4d  3d 0d 0a 4e 67 36 59 58   fdiAXgRM =..Ng6YX
    000005BD  62 77 67 2f 53 68 7a 42  4c 47 5a 38 52 36 6d 71   bwg/ShzB LGZ8R6mq
    000005CD  66 64 6a 75 74 4f 57 6c  45 38 48 6a 49 6a 69 56   fdjutOWl E8HjIjiV
    000005DD  44 51 65 2f 4a 49 3d 0d  0a 00                     DQe/JI=. ..

First Base64 Str: BZtfdiAXgRM=
  Decoded: 05 9B 5F 76 20 17 81 13

Second Base64 Str: Ng6YXbwg/ShzBLGZ8R6mqfdjutOWlE8HjIjiVDQe/JI=

Password: sapphire1 (will be encoded using unicode)
  Encoded: 73 00 61 00 70 00 70 00 68 00 69 00 72 00 65 00 31 00

sha256(First Base64 decoded + password encoded) =
sha256(05 9B 5F 76 20 17 81 13 73 00 61 00 70 00 70 00 68 00 69 00 72 00 65 00 31
00)
= 360e985dbc20fd287304b199f11ea6a9f763bad396944f078c88e254341efc92

base64_encode(360e985dbc20fd287304b199f11ea6a9f763bad396944f078c88e254341efc92) =
Ng6YXbwg/ShzBLGZ8R6mqfdjutOWlE8HjIjiVDQe/JI=  (matches second base64 str above,
password valid)
```

Authentication Bypass

```
SHA256 (server_nonce + base64_str + client_nonce)
```

HTTP Digest Auth

```
HA1 = MD5(username:realm:password)
HA2 = MD5(method:digestURI)
response = MD5(HA1:nonce:HA2)
```



tenable

Conclusion

- Use a popular, well supported cryptographic library in your projects rather than coming up with your own cryptographic functions. If possible, leave it as a shared library.
- For a complete solution for integrity, authentication, and confidentiality, use TLS 1.3. Use certificates for authentication rather than passwords.
- Don't use deprecated cryptographic routines functions.
- Encoding / obfuscation is not crypto.
- Use HMAC rather than MAC for integrity checking. Implement per RFC or use a library.
- Don't assume hard coded encryption keys in hardware can't be recovered. Even if you blow the security fuses.
- Store passwords properly as salted hashes.
- Look for prior work, and RFCs if you need help with some in particular.
- Have a few experts review your cryptographic implementations.

Password Authentication Best Practices - End Users

- Ideally random user IDs to prevent attackers guessing.

- Use *different* authentication solution for remote access than what is used internally (LDAP, AD, etc…). Ideally something hardened and designed to this purpose.
  - This solution use also utilize 2FA solution.

- Passwords should be at least 8 characters. OWASP recommends using a password "strength" meter rather than complexity requirements which can actually result in more predictable passwords.
  - https://github.com/zxcvbn-ts/zxcvbn

https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

tenable