mbed TLS
Adding Entropy Support

ARM

Ron Eldor
Security Staff Application Engineer
ARM mbed

Silicon Partner Workshop - Wyboston Lakes
March 2017

What is entropy / random?

- High entropy ≈ true random data
  - No information is available that can predict the future or tell about the past
  - Non deterministic

What is entropy / random?

DILBERT © 2001 Scott Adams.

What do these have in common?

2008:

Crippling Crypto: The Debian OpenSSL Debacle

2014:

2013:

RSA warns developers not to use RSA products

In today's news of the weird, RSA (a division of EMC) has recommended that developers desist from using the (allegedly) 'backdoored' Dual_EC_DRBG random number generator -- which happens to be the *default* in RSA's BSafe cryptographic toolkit. Youch.

RSA®
The Security Division of EMC

Bad entropy → Bad security

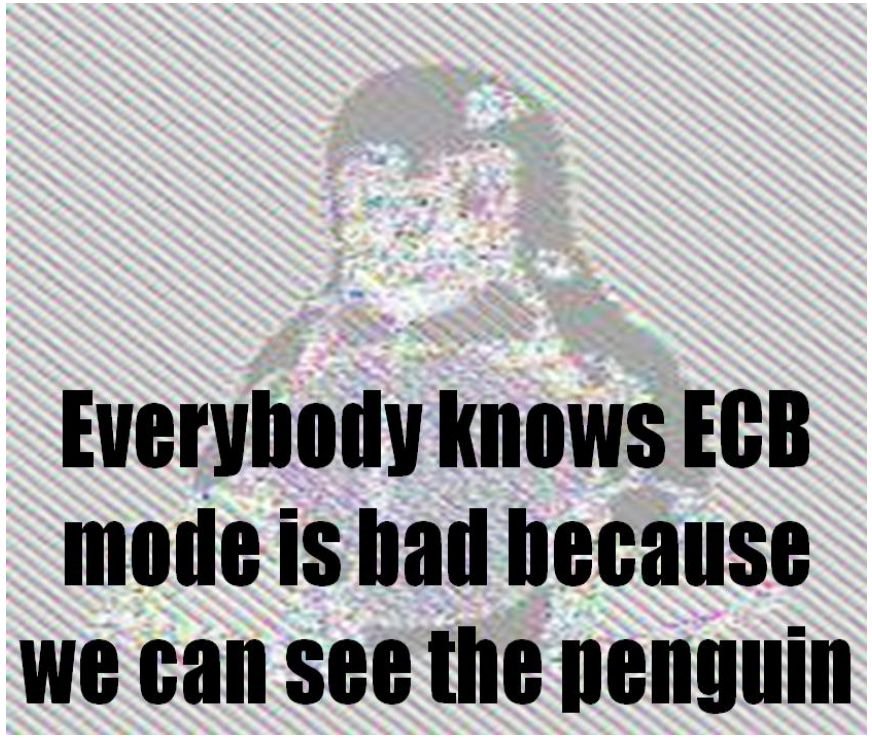
- Cryptography is as secure as its weakest link

Bad entropy → Bad security

- All cryptographic protocols depend on good entropy somewhere in the chain
  - Public/Private key pairs / identities (Generated with random)
  - Unique AES keys (Generated with random)
  - Unique IVs (Initialization Vectors)

  - And thus bad entropy also affects security of:
    - Session keys (e.g. SSL/TLS, SSH)
    - Encrypted files

- **The problem is that it still looks protected and encrypted to the eye**

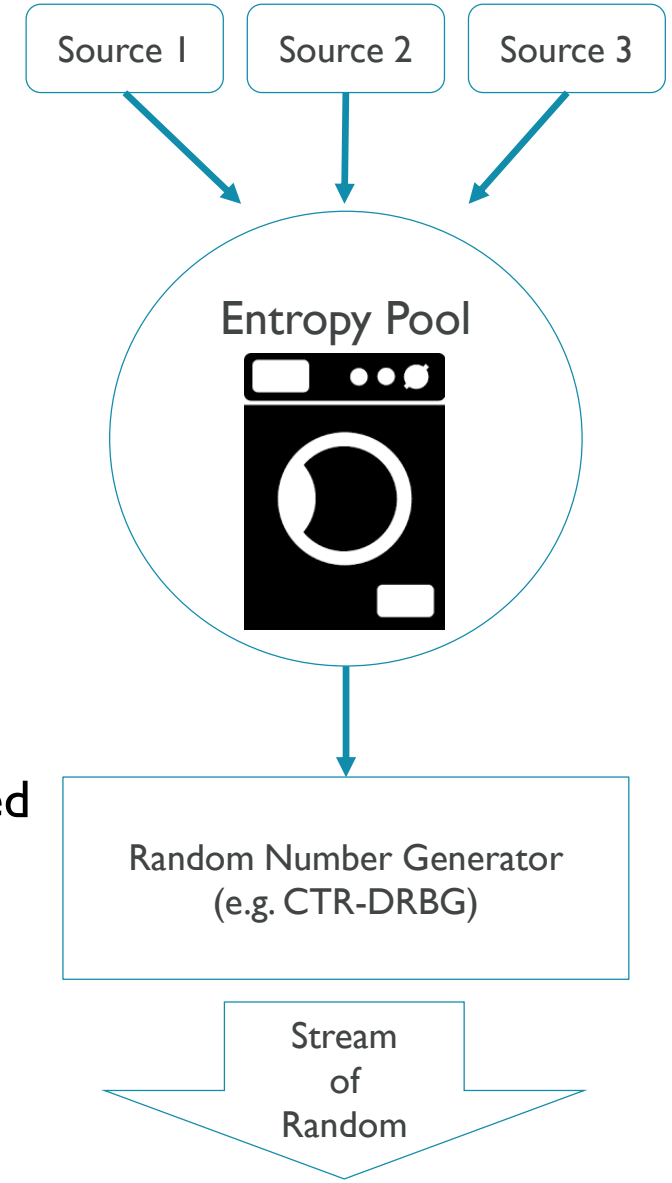


https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Implementing entropy collection within mbed TLS

How does entropy work in mbed TLS

- Entropy Sources
  - Provide a little bit of entropy
  - Can be strong or weak*
- Entropy Pool
  - Collects small entropy data
  - Jumbles them together into a really unpredictable number (the "seed")
- Random Number Generators (800-90A)
  - Starts from seed
  - Generates large number of non-deterministic bytes from seed
  - Seed should be regenerated regularly



* At least one strong Entropy source is required

The entropy quiz: Strong or weak?

- Hardware Clock / MCU tick counter?
  - Weak (Predictable)
- Seed compiled into binary image?
  - Weak (Does not change over boots, one leak breaks all devices)
- Timing / content of incoming network traffic?
  - Weak (manipulation/overview by attackers possible)
- Device-specific seed in non-volatile storage?
  - Reasonable (Changes over boot, but when known, it becomes predictable)

The entropy quiz: Strong or weak?

- HW Entropy source / TRNG in MCU?
  - Strong, based on HW environmental unpredictable noise

- By default mbed TLS does not know about strong entropy sources on targets!
  - Every target that has a strong entropy source needs to provide an entropy source function
  - HW Entropy to be implemented in targets/TARGET_XXXX

Implementation best practices

- Don’t allow unseen risks, have your security experts involved
- Use only one entropy pool in your system (!)
  - Initialized directly after boot with the proper sources for the target
- One global CTR-DRBG (synchronized) or one per thread (costs RAM)
- Add as many entropy sources as possible (the stronger the better)

Adding a strong entropy source (Compile-time)

- Add `TRNG` to device_has in targets.json
  https://github.com/ARMmbed/mbed-os/blob/master/targets/targets.json

- Implement the trng functions in mbed target:

```
void trng_init(trng_t *obj)
void trng_free(trng_t *obj)
int trng_get_bytes(trng_t *obj, uint8_t *output, size_t length, size_t *output_length)
```

  - `trng_t` is the trng context and will be defined in *objects.h*
  - `output` will be a pointer to the buffer the entropy pool expects to be filled with entropy
  - `length` will be the size of the buffer and thus the maximum size requested
  - `output_length` is to report back the amount of bytes provided by this HW poll function

Adding a strong entropy source (Compile-time)

- Real life example

```
void trng_init(trng_t *obj)
{
    (void)obj;
    CLOCK_EnableClock(kCLOCK_Rnga0);
    CLOCK_DisableClock(kCLOCK_Rnga0);
    CLOCK_EnableClock(kCLOCK_Rnga0);
}

void trng_free(trng_t *obj)
{
    (void)obj;
    CLOCK_DisableClock(kCLOCK_Rnga0);
}
```

Adding a strong entropy source (Compile-time)

- Real life example – cont.

```
int trng_get_bytes(trng_t *obj, uint8_t *output, size_t length, size_t *output_length)
{
    (void)obj;
    size_t i;

    RNG->CR = RNG_CR_INTM_MASK | RNG_CR_HA_MASK | RNG_CR_GO_MASK;

    for (i = 0; i < length; i++) {
        trng_get_byte(output + i);
    }

    if ((RNG->SR & RNG_SR_SECV_MASK) != 0) {
        return -1;
    }

    *output_length = length;

    return 0;
}
```

```
static void trng_get_byte(unsigned char *byte)
{
    size_t bit;

    for( bit = 0; bit < 8; bit++ )
    {
        while((RNG->SR & RNG_SR_OREG_LVL_MASK) == 0 );
        *byte |= (RNG->OR & 1) << bit;
    }
}
```

Replacing the entropy pool (runtime)

- The DRBG Seed APIs (e.g. `mbedtls_ctr_drbg_seed`) receive as a parameter the Entropy pool function:

```
int (*f_entropy)(void *, unsigned char *, size_t)
```

  - Use `mbedtls_entropy_func` by default
  - Replace with a different function, if TRNG should comply with a different standard
  - Parameters:
    - Context (mbedtls_entropy_context)
    - Output (The seed buffer)
    - Len (The seed required length)

What to do if no TRNG is available?

- When there is no hardware entropy source, you can choose to:
  - Use a NULL entropy source for testing (**unsecure!**)
    - No additional code required
    - Only suitable for evaluation
    - Never to be used in production
  - Add a non-volatile seed based entropy source
    - Requires two functions for the mbed TLS platform abstraction layer

Why NULL entropy

- Don’t use NULL entropy!!
- To be used for quick porting efforts
- To be used for evaluation
- Never for production

Enabling NULL entropy

- In order to use NULL entropy:
  - Explicitly add **all** entropy sources (MBEDTLS_HAVEGE_C, MBEDTLS_ENTROPY_HARDWARE_ALT, MBEDTLS_ENTROPY_NV_SEED) to *macros_remove* in *targets.json*
  - Enable NULL entropy in *targets.json* by adding to *macros*:
    - MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES
    - MBEDTLS_TEST_NULL_ENTROPY
- **Important**: (Run == work) != secure!

If the NULL ENTROPY feature is in place, entropy collection works, but no security is provided at all!

NV Seed entropy source

- When no TRNG is present on a target, there is no really good choice to provide entropy.
- A non-volatile seed entropy source provides a form of adding and updating a non-volatile seed in your device.
- This NV seed stored should be unique per device.
- The entropy source will request, expect and write 32 bytes in case of SHA-512 enabled (default), and 16 bytes if only SHA-256 is enabled

How to port NV Seed entropy source

- In order to use NV Seed entropy source:
  - Enable `MBEDTLS_ENTROPY_NV_SEED`

Summary: What needs to be done?

- Implement a target-specific entropy source and add the relevant macros to *targets.json*
  - Preferably poll the hardware entropy source (TRNG) available on the target
  - Otherwise NV seed entropy source might be an option

- Keep in mind:

  It's not hard to do entropy right, it's just very easy to do wrong

  - Might be valuable to work with the in-house security experts to have them do the implementation, confirm the right entropy sources, get implementation details or do a review

Hands-on workshop

- Use this branch:
  https://github.com/ARMmbed/mbed-os/tree/mbed-os-workshop-17q2

Questions?