Defeating TLS client authentication using fault attacks

Who are we ?

- R&D dept. @ Kudelski Security
- Embedded systems security research
  - Reverse engineering
- Security Evaluation Lab @ Kudelski IoT
- Hardware attacks
  - Glitch / EM
  - Lasers !

Introduction

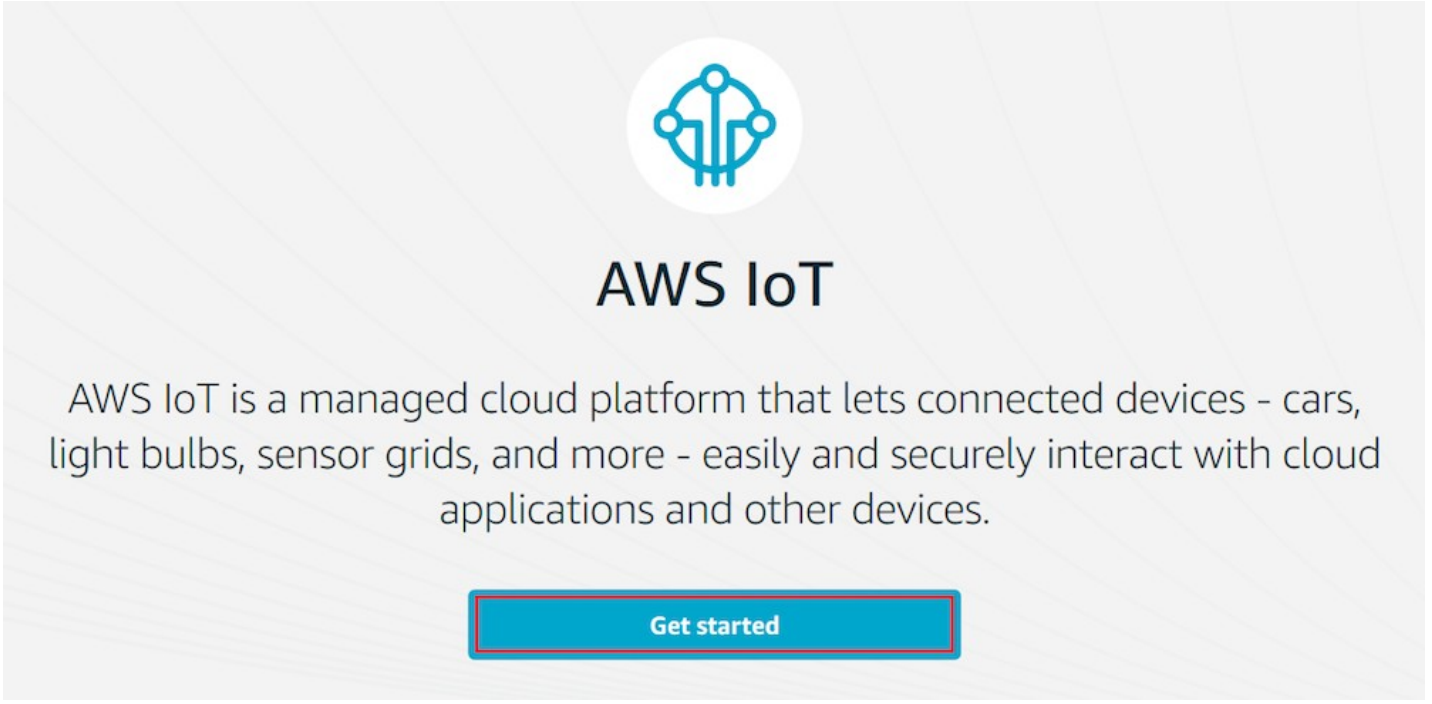
- In more and more use cases we have an embedded device which communicates with a cloud.
- The device (usually) authenticates itself to guarantee data origin.
- Some of the devices are low cost and have no physical security.

TLS (in a nutshell)

TLS 1.2

- **Transport Layer Security** replaces **Secure Sockets Layer**
- De facto standard (the **s** of https and green lock in the browser)
- Current version is TLS 1.3 released in 2018.
- TLS 1.2 is still massively used.
- Used in IoT for mutual authentication with the cloud.

Amazon Web Services IoT



AWS IoT authentication

- TLS 1.2 authentication is used by AWS IoT to identify devices.
- AWS FreeRTOS uses mbedTLS from ARM to implement TLS.

- **AWS IoT cloud supports the following cipher suites:**

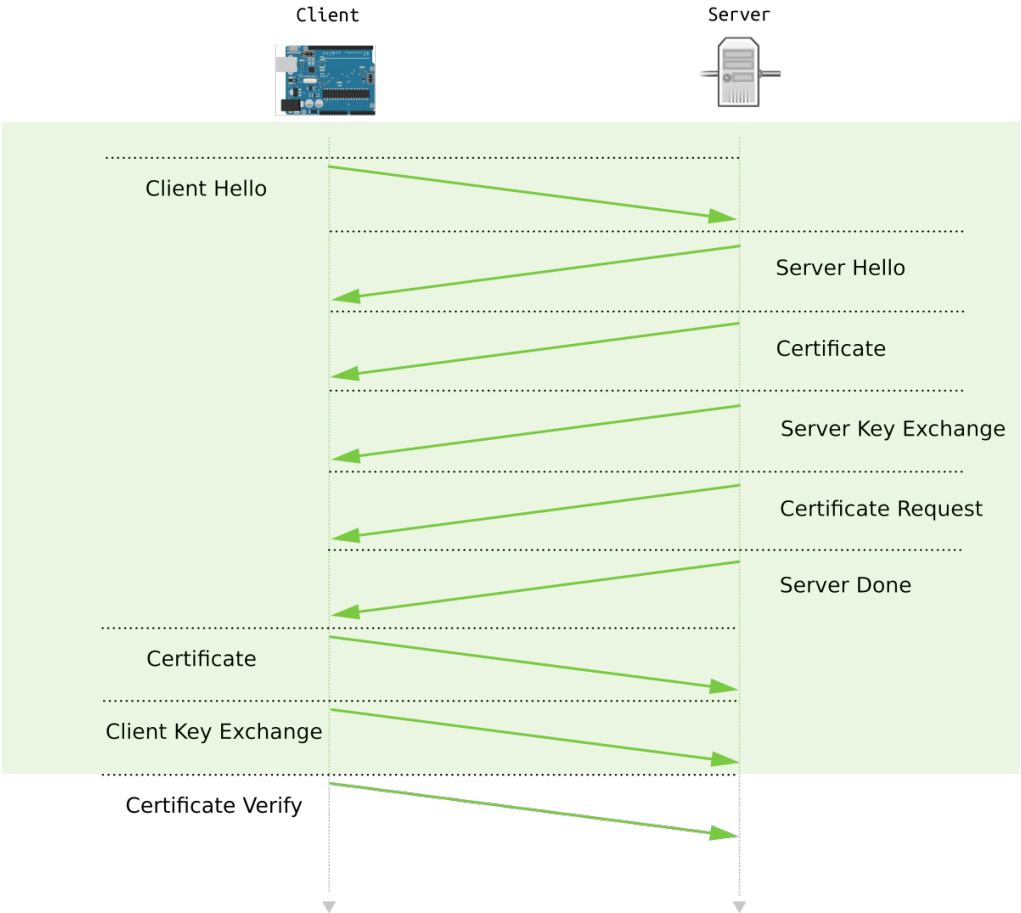
ECDHE-ECDSA-AES128-GCM-SHA256 (recommended)
ECDHE-RSA-AES128-GCM-SHA256 (recommended)
ECDHE-ECDSA-AES128-SHA256
...

AWS IoT authentication

Client certificate and private key are in the firmware:

```
/*
 * PEM-encoded client private key.
 *
 * Must include the PEM header and footer:
 * "-----BEGIN RSA PRIVATE KEY-----\n"\
 * "...base64 data...\n"\
 * "-----END RSA PRIVATE KEY-----\n"
 */
#define keyCLIENT_PRIVATE_KEY_PEM \
"-----BEGIN EC PRIVATE KEY-----\n"\
"MHcCAQEEII4UCIb5bvJp5zbLH+JOoSYn32lx8y1sFbbLwoEK8ZOCoAoGCCqGSM49\n"\
"AwEHoUQDQgAEOns7GCQTAY1QYVi4z83GY0r+/8+FkGSP/NtP0YK8kfqYRMeqGPBB\n"\
"PPhFQOfiYF7oSKE74qzCp4VpBwbvjbTTSw==\n"\
"-----END EC PRIVATE KEY-----\n"
```

TLS 1.2 handshake



Certificate Verify signature

Elliptic Curve Digital Signature Algorithm (in a nutshell too)

ECDSA signature

- TLS allows using RSA or ECDSA as signature algorithms.
- ECDSA has the advantage to have smaller key lengths for the same security level.
- Performance of ECDSA is better for signature.

Perfect signature algorithm for IoT.

ECDSA

From **d**, the device private key, the signature is computed over the elliptic curve:

$$(x, y) = k \cdot P$$
$$r = x$$
$$s = k^{-1}(h + rd)$$

The output signature is **(r, s)**.

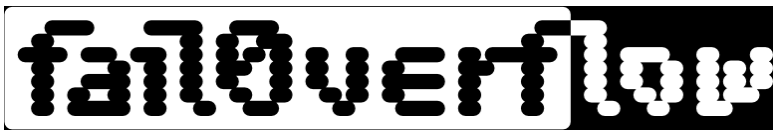
The nonce **k** must be generated randomly and must be unique.

ECDSA attack

From **r** it is not possible to recover the value of **k** (discrete logarithm).
But if two different messages have been signed with the same nonce then it is possible to recover **k**.

Then with **k** we can recover **d** the private key directly.

2010: PS3 signature key recovery



Fault attacks on ECDSA

If we are able to set the nonce to a known value or to reduce its entropy then the private key can be recovered with:

$$d = (ks - h) \cdot r^{-1}$$

h is known since it is the hash of previous handshake messages

MbedTLS implementation

mbedTLS

Used in a lot of embedded SDKs

“**mbedTLS offers an SSL library with an intuitive API and readable source code, so you can actually understand what the code does.**” (tls.mbed.org)

We analyzed the code until we reached the nonce generation

MbedTLS implementation

The nonce is generated in the mbedtls_ecp_gen_privkey in ecp.c:

```
#if defined(ECP_SHORTWEIERSTRASS)
    if( ecp_get_type( grp ) == ECP_TYPE_SHORT_WEIERSTRASS )
    {
        /* SEC1 3.2.1: Generate d such that 1 <= n < N */
        int count = 0;

        /*
         * Match the procedure given in RFC 6979 (deterministic ECDSA):
         * - use the same byte ordering;
         * - keep the leftmost nbits bits of the generated octet string;
         * - try until result is in the desired range.
         * This also avoids any biais, which is especially important for ECDSA.
         */
        do
        {
            MBEDTLS_MPI_CHK( mbedtls_mpi_fill_random( d, n_size, f_rng, p_rng ) );
            MBEDTLS_MPI_CHK( mbedtls_mpi_shift_r( d, 8 * n_size - grp->nbits ) );
```

MbedTLS implementation

```c
/*
 * Import X from unsigned binary data, big endian
 */
int mbedtls_mpi_read_binary( mbedtls_mpi *X, const unsigned char *buf, size_t buflen )
{
    int ret;
    size_t i, j;
    size_t const limbs = CHARS_TO_LIMBS( buflen );

    MPI_VALIDATE_RET( X != NULL );
    MPI_VALIDATE_RET( buflen == 0 || buf != NULL );

    /* Ensure that target MPI has exactly the necessary number of limbs */
    if( X->n != limbs )
    {
        mbedtls_mpi_free( X );
        mbedtls_mpi_init( X );
        MBEDTLS_MPI_CHK( mbedtls_mpi_grow( X, limbs ) );
    }

    MBEDTLS_MPI_CHK( mbedtls_mpi_lset( X, 0 ) );

    for( i = buflen, j = 0; i > 0; i--, j++ )
        X->p[j / ciL] |= ((mbedtls_mpi_uint) buf[i - 1]) << ((j % ciL) << 3);
```

MbedTLS random nonce generation

Fill a buffer with random values:
  Depending on SDK/target, will use the hardware RNG
Convert the buffer to a mbedtls_mpi value:
  Converts buffer from big- to little-endian...
    By copying the buffer bytes to dword

Attack Idea

- Generate a lookup table containing small nonces multiplied by the generator *i.e.* with records **(k, k · P)**.
- Insert a fault to exit the buffer copy loop earlier.
- The resulting nonce value may be truncated (32 bits).
- If the resulting signature is in our table then we can recover the nonce and then private key !

Code protection ?

- Return value is uninitialized at the beginning of the function
  - Compiler initializes the value to 0...
  - Function returns 0 if successful

Exploitation

ESP32

- System-on-Chip manufactured by Espressif
- Widely deployed on the field
- Supported by AWS IoT
- Integrated Wi-Fi
- Vulnerable to voltage glitch

Previous fault attacks on ESP32

- LimitedResults:
  - Voltage glitch
  - Effects used to
    - Bypass AES encryption
    - Bypass secure boot
    - Extract flash encryption and secure boot keys

ESP32 power domains

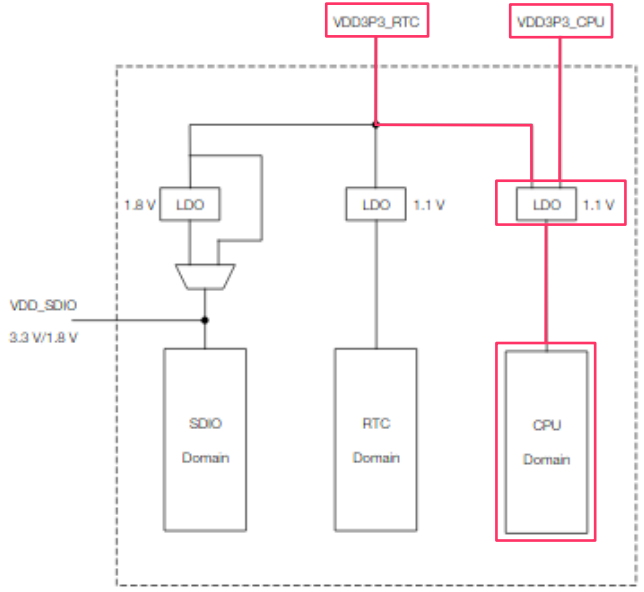
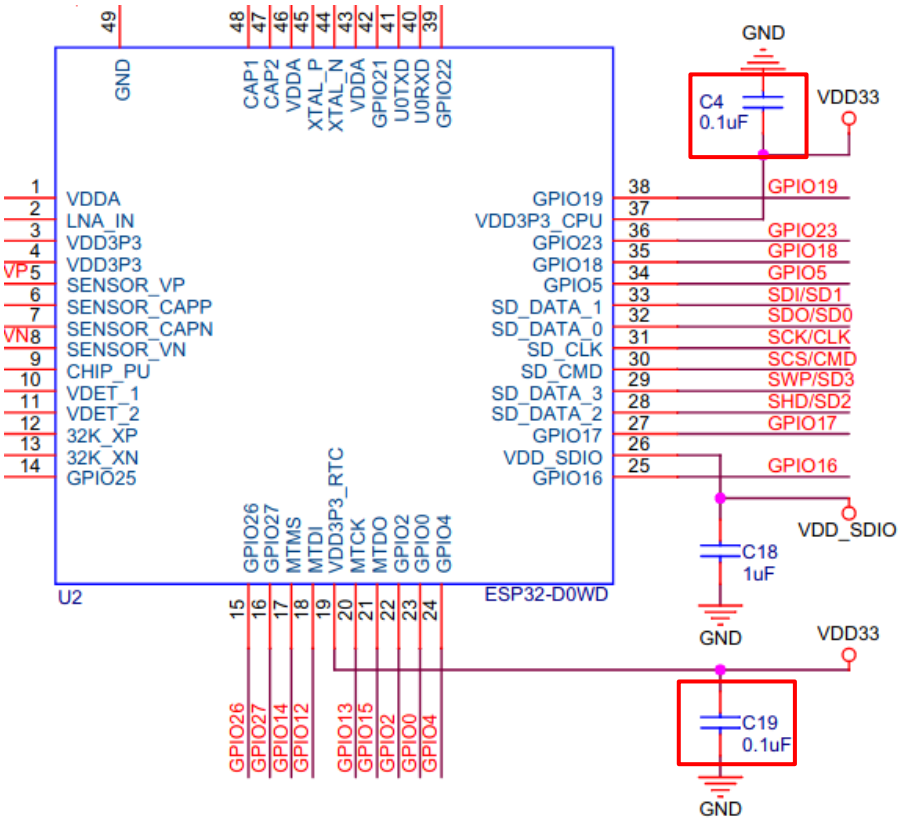
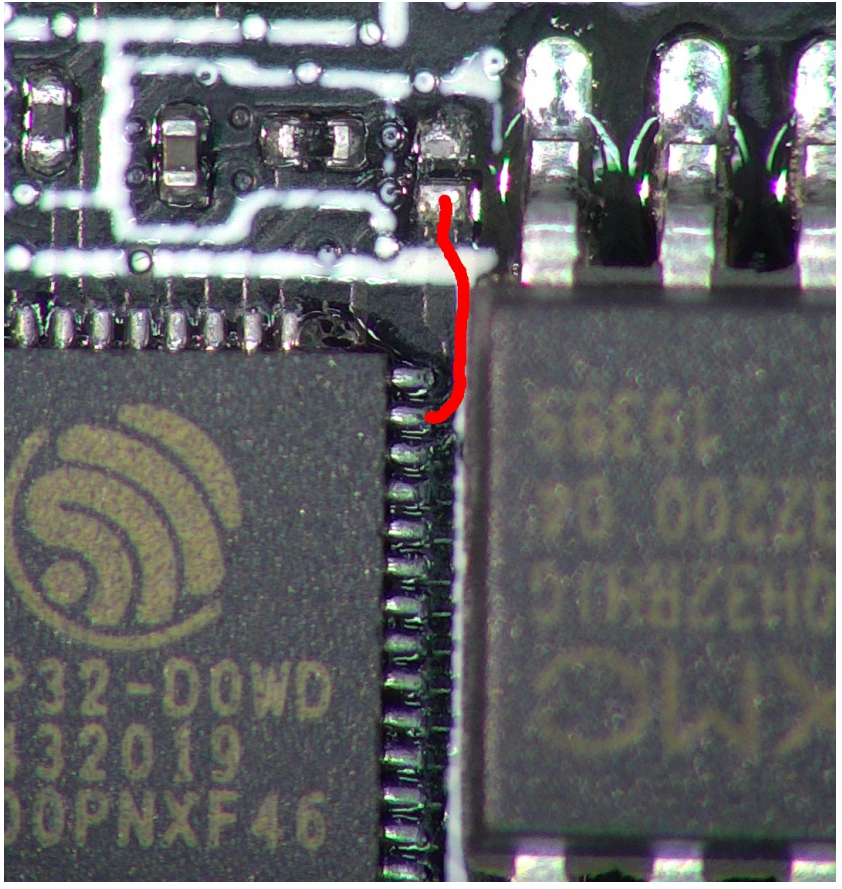


Figure 4: ESP32 Power Scheme

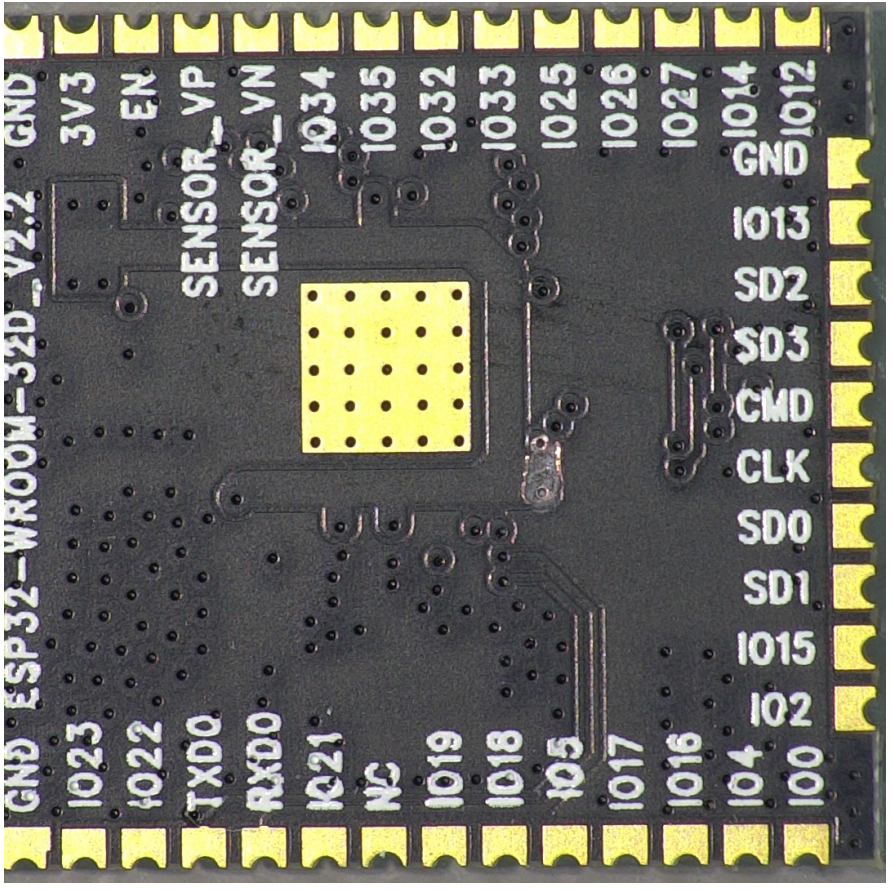
Voltage glitch on ESP32



ESP32 preparation

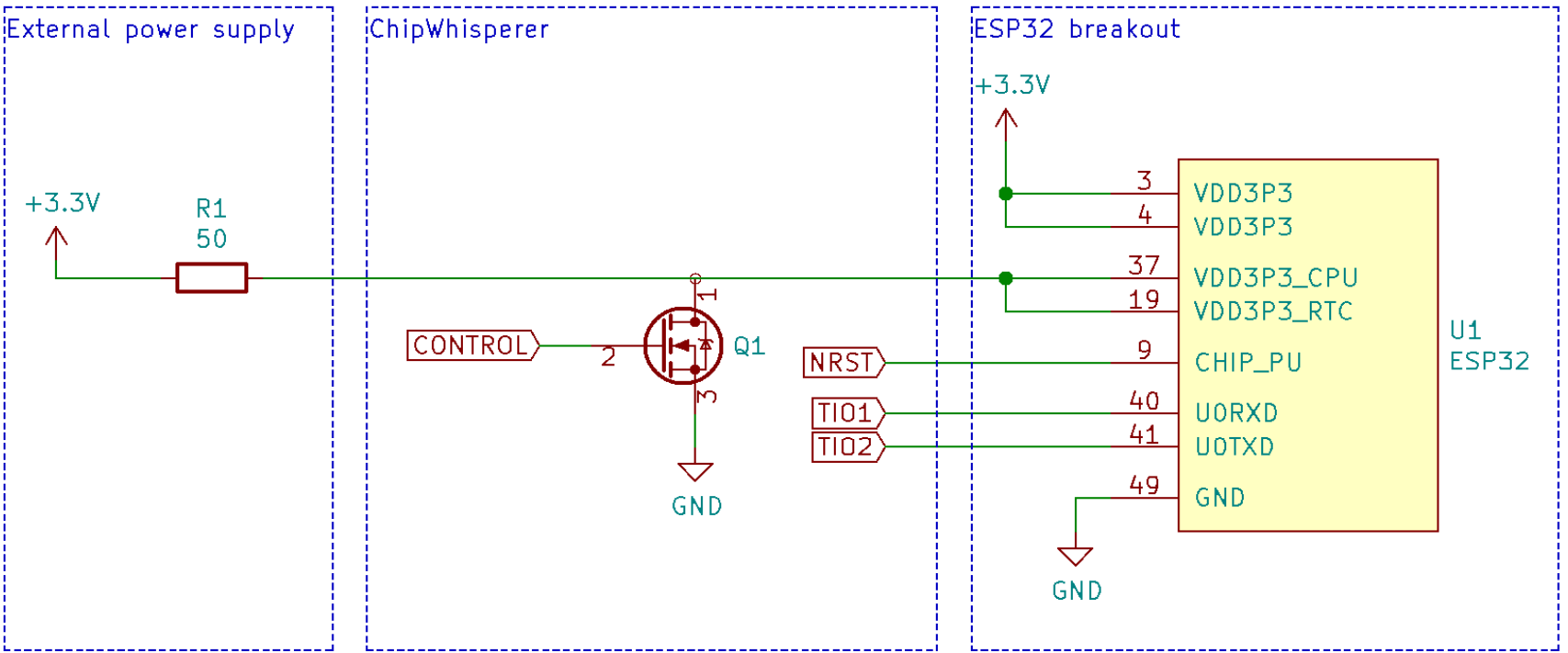


ESP32 preparation

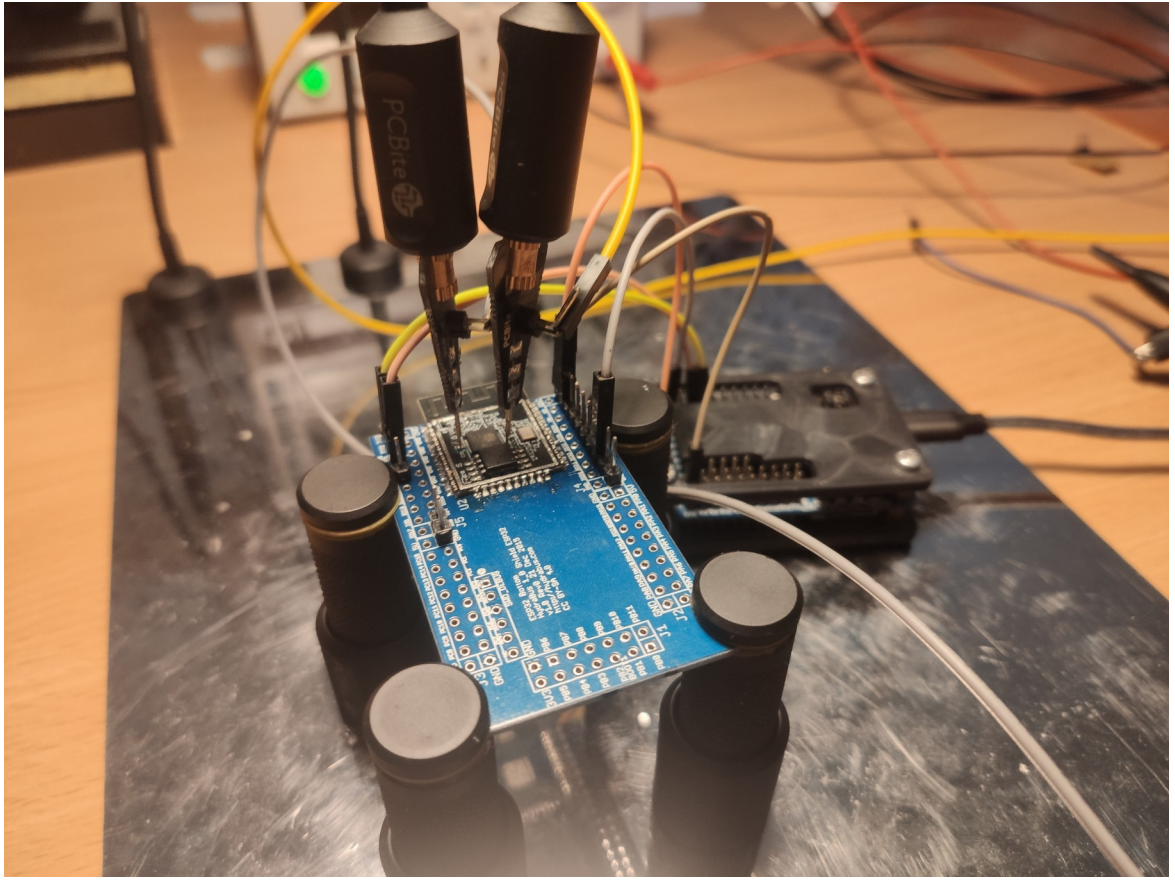


Chipwhisperer setup

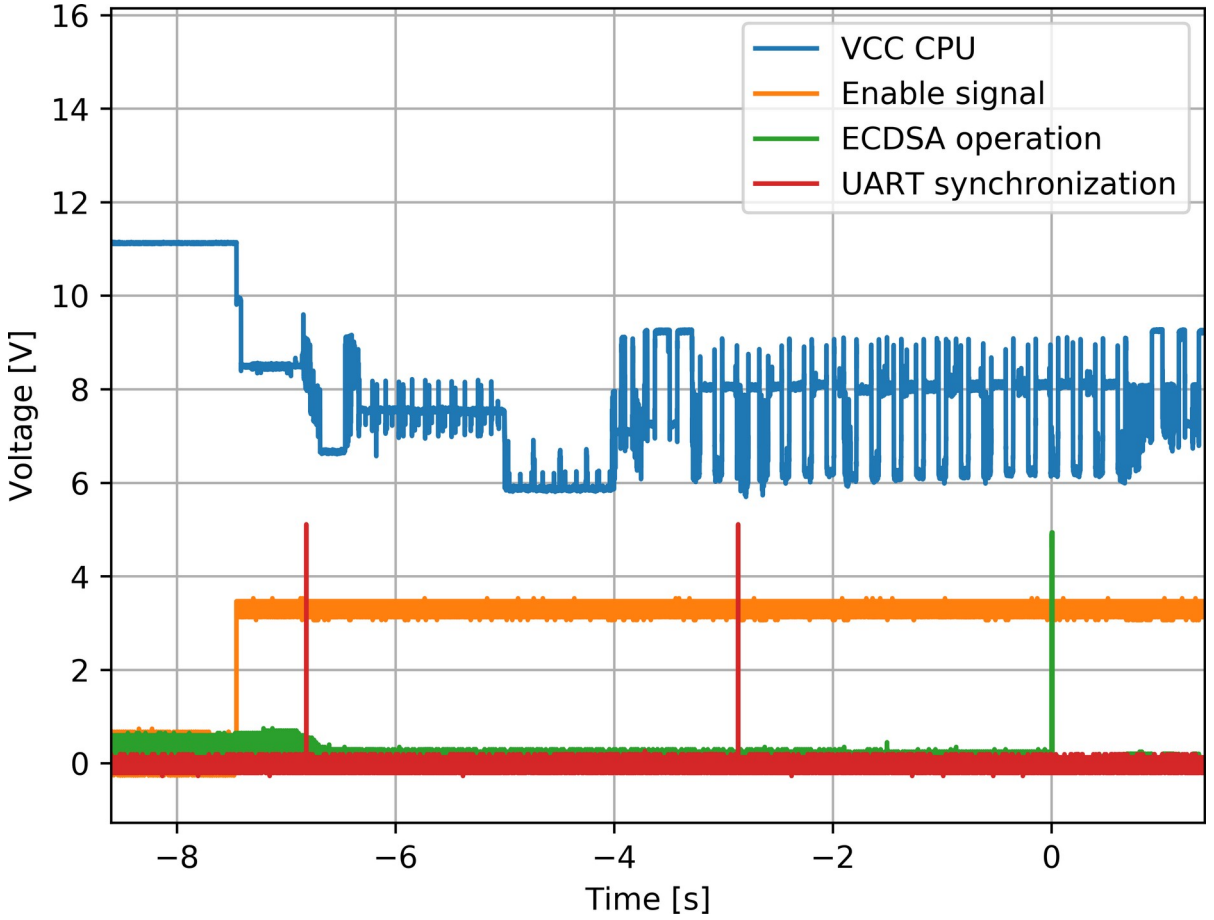
- Voltage glitch was generated by Chipwhisperer using crowbar method.



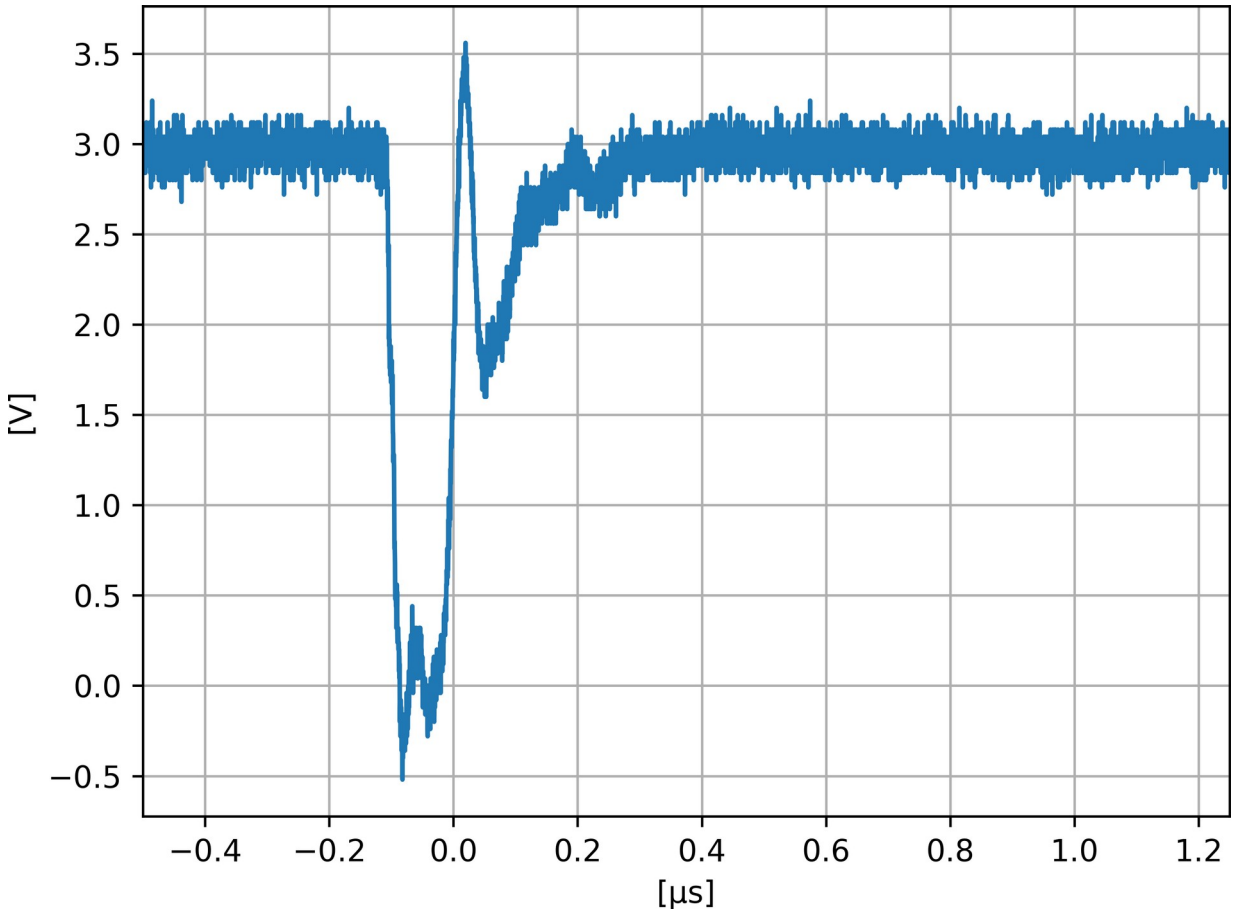
Chipwhisperer setup



ESP32 start-up



Glitch shape



Lookup table

- We generated a lookup table for **k** from 1 to 2^{32}, around 300GB.
- It took two days to generate the table but then one lookup takes 5s.
- The table is similar to the one used during an attack against Bitcoin signature. (https://github.com/nomeata/secp265k1-lookup-table.)

Key recovery

- The network was probed and each signature was recorded with the corresponding hash of the previous handshake messages
- If the signature is in our database:

Quick win

- During the tests, we found a glitch point that fixes the nonce to 0xFFFFFFFF
- Eases the cracking process

Disclosure

MbedTLS implementation

- The call to **read_binary** was removed from version after 2.16.1 of mbedTLS for performance reasons. But it was still included in ESP32 software until February 2020.
- (Un)fortunately, there are other ways to attack the signature with the same results (CTR_DRBG or HMAC_DRBG).

MbedTLS implementation

```
    /*
     * Set initial working state.
     * Use the V memory location, which is currently all 0, to initialize the
     * MD context with an all-zero key. Then set V to its initial value.
     */
    if( ( ret = mbedtls_md_hmac_starts( &ctx->md_ctx, ctx->V,
                                        mbedtls_md_get_size( md_info ) ) ) != 0 )
        return( ret );
    memset( ctx->V, 0x01, mbedtls_md_get_size( md_info ) );

    if( ( ret = mbedtls_hmac_drbg_update_ret( ctx, data, data_len ) ) != 0 )
        return( ret );

    return( 0 );
}
```

Disclosure

- We contacted ARM with full details of our attack.
- We suggested to change the default return value to something else in our responsible disclosure
- About one month later :

> *“[...]**We generally consider hardware fault attacks out of scope of the Mbed TLS threat model.** However, we are happy to work with you on this issue and follow coordinated disclosure with the fix.*

- No more communication from ARM since then

No response ?

Initialise return values to an error

Browse files

Initialising the return values to and error is best practice and makes the library more robust.

development (#3085) mbedtls-2.22.0

yanesca committed on Nov 22, 2019 — 1 parent a13b905 commit 24eed8d2d2df4423a63c8761edd0d65a43ff03a3

Showing 43 changed files with 322 additions and 279 deletions. Unified Split

35 library/bignum.c

```
@@ -46,6 +46,7 @@
 #include "mbedtls/bignum.h"
 #include "mbedtls/bn_mul.h"
 #include "mbedtls/platform_util.h"
+#include "mbedtls/error.h"

 #include <string.h>

@@ -314,7 +315,7 @@ int mbedtls_mpi_safe_cond_swap( mbedtls_mpi *X, mbedtls_mpi *Y, unsigned char sw
  */
 int mbedtls_mpi_lset( mbedtls_mpi *X, mbedtls_mpi_sint z )
 {
-    int ret;
+    int ret = MBEDTLS_ERR_ERROR_CORRUPTION_DETECTED;
     MPI_VALIDATE_RET( X != NULL );
```

Timeline

- 09/09/2019 : Vulnerability reported to ARM.
- 09/27/2019 : ARM acknowledge the vulnerability.
- 11/22/2019 : ARM hardened the library with error status.
- 02/12/2020 : Espressif upgraded to mbedTLS v2.16.5.
- Now : Vulnerability still exists.

Possible countermeasures

- Use TLS 1.3
  - Handshakes are encrypted
- Use RSA for authentication ?
- Use a hardware secure element

Conclusions

Takeaways

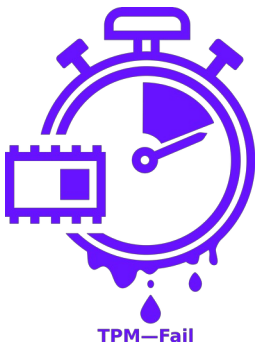
- Full key recovery is possible using a single fault.
- This attack is not related to the target platform.
- Software hardening must be implemented carefully.

Questions ?

Backup slides

Previous attacks on ECDSA

- **2014: “Ooh Aah... Just a Little Bit”**
- **2019: Biased Nonce Sense: Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies**
- **2019: TPM.fail**
- **2019: Minerva**



Previous fault attack on TLS

- **Attacking Deterministic Signature Schemes using Fault Attacks** *(Poddebniak et al.)***:**
  - Rowhammer on deterministic ECDSA and EdDSA.
  - Server attack.
  - Needs one faulted and one correct signature for the same message.
- **Degenerate Fault Attacks on Elliptic Curve Parameters in OpenSSL** *(Takahashi et al.)***:**
  - Fault attack on point decompression.
  - Application on OpenSSL running on Raspberry Pie.

Degenerate Fault Attacks

mpg commented on Feb 7, 2019 Contributor

Support for compressed format has been deprecated by RFC 8422 in the context of TLS, which reflects a more general sentiment in the ECC community to prefer uncompressed format. Also, implementing it correctly for all supported curves would require substantial code, impacting our footprint - and the present PR would require non-trivial rework (values of P not congruent to 3 mod 4, unit tests) before if would be ready for merge.

At this point, we're unlikely to want to add that amount of code for a feature that's formally deprecated in TLS and being abandoned more generally, so I'm closing this PR.

Thanks for your contribution and interest in Mbed TLS anyway.

 **mpg** closed this on Feb 7, 2019

ESP32 preparation