One&Done: A Single-Decryption EM-Based Attack on OpenSSL’s Constant-Time Blinded RSA

Monjur Alam, Haider Adnan Khan, Moumita Dey, Nishith Sinha, Robert Callan, Alenka Zajic, and Milos Prvulovic
**Motivation**

- Public key crypto is essential for modern security
  - Secure exchange of session keys
  - Verifying identity of systems and users
  - And a lot more
- Private keys are a highly valuable asset
  - So attackers want them
  - And we don’t want attackers to get them
Public Key Crypto

- Good public key crypto (e.g. RSA)
  - Designed to make private keys very, very hard to recover
Analog Side-Channel Attacks

- But cryptographic implementation runs on real hardware
  - Logic gates switch, causing current flow
  - Currents flowing create changes in surrounding EM field

Key

RSA

Side-channel information helps recover the private key
Analog Side-Channel Attacks

- Message randomization (blinding)
  - Prevents chosen-plaintext and other message-dependent attacks
- But… when message-independent operations use the key alone, eventually enables efficient recovery of the private key
Analog Side-Channel Attacks

- **One&Done**
  - Message does not matter (message blinding does not help)
  - Multiple “traces” not needed (exponent blinding does not help)

Side-channel information **alone**, in a **single encryption/signing**, enables efficient recovery of the entire private key.
OpenSSL’s RSA Implementation

- BN_mod_exp_montgomery_consttime()
  - Computes $x^d \mod m$, where $d$ is the secret exponent

```c
b=bits - 1;
while (b>=0){
    wval=0;
    // Scan the window,
    // squaring the result as we go
    for (i=0;i<w;i++) {
        BN_mod_mul(v,v,v,m);
        wval<<=1;
        wval+=BN_is_bit_set(d,b);
        b--;
    }
    // Multiply window’s result
    // into the overall result
    BN_mod_mul(v,v,ct[wval]);
}
```

- For each fixed-size “window”
- For each bit in the window
- Square the result ($v=v^2$)
- Look up one bit of $d$ and add to $wval$
- Multiply result with $x^{wval}$
- Look up precomputed $x^{wval}$
Side-Channel Attacks on OpenSSL’s RSA

- **BN_mod_exp_montgomery_consttime()**
  - Computes $x^d \mod m$, where $d$ is the secret exponent

  ```
  b = bits - 1;
  while (b >= 0) {
    wval = 0;
    // Scan the window,
    // squaring the result as we go
    for (i = 0; i < w; i++) {
      BN_mod_mul(v, v, v, m);
      wval <<= 1;
      wval += BN_is_bit_set(d, b);
      b--;
    }
    // Multiply window's result
    // into the overall result
    BN_mod_mul(v, v, ct[wval]);
  }
  ```

  - For each fixed-size “window”
  - For each bit in the window
  - Square the result ($v = v^2$)
  - Get bit from $d$, add to $wval$
  - Multiply result with $x^{wval}$
  - Look up precomputed $x^{wval}$

Genkin et al., CHES’15

- **One&Done (new)**
- **Mitigation (new)**

Message Blinding

Cache (e.g. Percival)

Scatter-Gather
Measurement Setup

Samsung Galaxy Centura SCH-S738C
Alcatel Ideal
A13-OLinuXino
Side Channel Analysis

Recent advances in side-channel-based program monitoring

- Camelia, our DARPA LADS project
  - Uses analog signals to monitor computational activity to detect control flow deviation and/or execution of unknown code
  - Found that even a single-instruction control-flow can be detected
  - But…

Constant-time implementation – no key-dependent CF

- Every encryption has the same CF sequence
  - Can’t use CF differences for attack
  - But can use the (very stable and predictable) signal features and timing to tell us exactly where in the signal BN_is_bit_set is executing
**Attack Approach**

Constant-time Montgomery Multiplication to square the result

Another Constant-time Montgomery Multiplication

**Easy to Find**

```
6    for (i=0; i<w; i++) {
7      BN_mod_mul(v, v, v, m);
8      wval<<=1;
9      wval+=BN_is_bit_set(d, b);
10     b--;
11   }
```
Relevant Part Zoom-In
How well does this recover bits of \(<d_p,d_q>\)?

- Training on 15 private-key RSA decryptions
- Recover bits of secret exponents using only **one** decryption

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>Max</td>
</tr>
<tr>
<td>99%</td>
<td>Max</td>
</tr>
<tr>
<td>99%</td>
<td>Max</td>
</tr>
<tr>
<td>98%</td>
<td>Max</td>
</tr>
<tr>
<td>98%</td>
<td>Median</td>
</tr>
<tr>
<td>97%</td>
<td>Median</td>
</tr>
<tr>
<td>97%</td>
<td>Median</td>
</tr>
<tr>
<td>96%</td>
<td>Min</td>
</tr>
<tr>
<td>95%</td>
<td>Min</td>
</tr>
</tbody>
</table>

- Samsung Galaxy Centura Phone
- Alcatel Ideal Phone
- OLinuXino Board
Full RSA Key Recovery

- We have $d_p$ and $d_q$ but with
  - Erasures – could not find where the bit’s signal is
  - Errors – found the bit’s signal, but misclassified it (0 vs. 1)

- Existing branch-and-prune algorithms
  - Prune partial solutions when group of bits has too many errors
    - Assumes errors are uniformly distributed
    - Our errors often occur in bursts
    - Does not explicitly handle erasures
  - Prune partial solutions that disagree with known bits of $<d_p,d_q>$
    - Can’t handle errors (no bits truly “known”)

Full RSA Key Recovery

- We have dp and dq but with
  - Erasures – could not find where the bit’s signal is
  - Errors – found the bit’s signal, but misclassified it (0 vs. 1)

- Our algorithm
  - Take partial solution with fewest disagreement overall
    - Known-to-be-unknown bits (erasures) not counted
  - Expand that partial solution by one bit position
    - Prune expansions that violate relationships between p, q, n, dp, and dq
    - Efficient implementation, nearly all checks use only scalars (not BNs)
  - Repeat
Recover RSA key from $\langle d_p, d_q \rangle$ with errors

- Key search using one i7 core: 500K steps / second!
More in the paper

- Train on one device, attack another
  - Only slightly worse than same-device (still 100% key recovery)
- Similar attack on sliding-window implementation
  - Used in prior versions of OpenSSL
    - Prior attacks extract enough bits to sometimes allow full-key recovery
  - One&Done recovers nearly all bits in one private-key encryption, recovered full key every time
Mitigation

- Fundamental enabler of the attack
  - Several instructions have very few possibilities for their operands
    - BN_is_bit_set returns either 0 or 1
- No need to get bits one at a time
  - A 5-bit fixed window needs 5 consecutive bits
    - Don’t have to get them one at a time and shift into wval
  - So we take an entire word’s worth of bits each time, mask to window-size only before wval is needed
  - Takes only a little longer than getting one bit!
  - But done only once per window!
Results after mitigation

![Graph showing results after mitigation]

- Random Guessing:
  - Min: 40%
  - 45%
  - 50%
  - 60%
  - 65%

- Erasures Counted as Errors:
  - Min: 40%
  - 45%
  - 50%

- Devices:
  - Samsung Galaxy Centura Phone
  - Alcatel Ideal Phone
  - OLinuXino Board

The graph illustrates the results after mitigation, with random guessing and erasures counted as errors for different devices.
Conclusions

- Analog side-channel attack on OpenSSL’s constant-time modular exponentiation implementation
  - Precise timing thanks to constant-timeness of the implementation
  - Highly accurate thanks to one-secret-bit-at-a-time implementation
- Entire private key recovered from only one use of that key
- Attack not affected by blinding
  - Attack directly obtains exponent bits, message bits not relevant
  - Exponent blinding does not help against single-trace attacks
- Mitigation: look up groups of secret bits, not individual bits
Thank you!

Questions?