ECDSA's Low-Cost (but) High-Security Solution

**What & Why?**

* Side-Channel Analysis (SCA) attacks pose a significant threat to the security of cryptographic implementations because they exploit some information (e.g., power consumption or electromagnetic radiation) that can be observed from the execution of an algorithm on a device to recover the secret key.
* SCA attacks can be divided into two main groups: single-trace and multi-trace attacks.
* Countermeasures try to reduce the correlation between the targeted secret and captured side-channel measurement, while it also brings overhead on the design.
* Although single trace attacks can be overcome with lightweight countermeasures, it is costly to mitigate multi-trace attacks.
* A formal approach to mitigate multi-trace side-channel attacks is to mask the implementation.
* Masking has a considerable overhead, affecting area, power consumption, latency, and/or throughput by a factor of two or three at least.
* Our solution relies on a comprehensive analysis of an efficient implementation of ECDSA.
* Based on this comprehensive analysis, we created a threat model where we define the single-trace and multi-trace attack points.
* Rather than simply masking all implementation, we mask necessary locations and provide a low-cost solution with high security.

**﻿How & Details?**

**ECDSA Background & Side Channel issues**

* ECDSA: ECDSA (Elliptic Curve Digital Signature Algorithm) engine offering a variant of the cryptographically secure Digital Signature Algorithm (DSA) which uses elliptic curve (ECC). A digital signature is an authentication method used where a public key pair and a digital certificate are used as a signature to verify the identity of a recipient or sender of information. This digital signature algorithm has three main routines: key generation, signing and verifying.
  + **Key generation Algorithm (KeyGen)**
    - This routine has two steps of operations. First it employs HMAC DRBG engine to generate a private key. The private key is then used to generate a public key.
    - However, the public key generation operation consists of multiplication, called scalar multiplication. The multiplication’s operand are private key and a known value.
    - Intuitively, KeyGen does not posses a threat for multi-trace attacks. There are single-trace including simple-power-analysis (SPA) and horizontal power attacks.
  + **Signing Algorithm**
    - While it includes the same steps with KeyGen, it has additional operations.
    - The additional operations are modular integer multiplications and additions.
    - The modular multiplications and additions possess multiple side-channel threats
  + **Verifying Algorithm**

This routine works with public values and thus it does not possess a SCA attack threat.

* The summary of analysis is given with the following tables where the columns show the operation, its vulnerability, and its protection against the described vulnerability.

|  |  |  |
| --- | --- | --- |
| **KeyGen Algorithm** | | |
| Operation | Vulnerable to Attack | Protection Method |
| Private Key = HMAC DRBG | X | X |
| Publick Key = Private Key \* G | 1. Timing Attacks 2. SPA 3. Horizontal SCA | 1. Montgomery Ladder Multiplication 2. Scalar Blinding 3. Scalar Blinding |

|  |  |  |
| --- | --- | --- |
| **Signing Algorithm** | | |
| Operation | Vulnerable to Attack | Protection Method |
| h= HASH (message) | X | X |
| Ephemeral Key (k) = HMAC DRBG | X | X |
| r = Ephemeral Key × G | 1. Timing Attacks 2. SPA 3. Horizontal SCA | 1. Montgomery Ladder Multiplication 2. Scalar Blinding 3. Scalar Blinding |
| s = k-1  \* (h + r \* private Key) | 1. Timing Attacks 2. SPA 3. Horizontal SCA 4. CPA, DPA | 1. Lightweight Masking 2. Lightweight Masking 3. Lightweight Masking 4. Lightweight Masking |

**Solution:**

* Our solution uses an efficient masking approach to address the issue. Masking randomizes the intermediate values of an implementation by splitting them in two or more shares that are processed independently.
* In ECDSA algorithm, there are two secret values, k and private key, that perform the signing routine. If we provide masking using both k and private key, then the design cost would approximately increase by at least 10x in modular integer operations, which in turn costs increase in area, power and potentially making the design’s timing convergence worse.
* Therefore, we mask only private Key to provide masking level security for all modular integer operations and reuse the base design by only changing the sequencer of the design. Diagram

  Description automatically generated
* We split only private Key as P1= private Key – d and P2 = d where d is a random number ranging between 0 and 2384 .
* Then, the last operation of signing is as follows both for masked version and unprotected version

|  |  |  |
| --- | --- | --- |
| Share 1 | Share 2 | Unprotected |
| PR1= r\*P1 | PR2= r\*P2 | PR = r\* privKey |
| PRH1 = PR1+h | PRH2 = PR2+h | PRH = PR+h |
| s1 = k-1  \* PRH1 | s2 = k-1  \* PRH1 | s = k-1  \* PRH |

* The table above shows that our solution increases the complexity of the modular operation by only two times.
* This protection does not require additional hardware resources. It increases the execution time of signing step, with a minimal overhead on the ECDSA.
* For the random number generation, we reuse already existing HMAC DRBG that is required as a part of ECDSA engine.