CSE 5095 & ECE 4451 & ECE 5451 – Spring 2017
Lecture 9a

- RSA part of the Slide deck originally based on some material by Chenglu during ECE 6095 Spring 2017 on Secure Computation and Storage, a precursor to this course
- SHIELD part follows http://eprint.iacr.org/2015/210

RSA Background and Timing Attack Secure and Efficient Initialization and Authentication Protocols for SHIELD

Marten van Dijk
Syed Kamran Haider, Chenglu Jin, Phuong Ha Nguyen

Department of Electrical & Computer Engineering
University of Connecticut

UCONN

Secure Computation Laboratory

RSA Background

RSA Background

- RSA: parameters
- 1. Pick two random primes, p and q. Let n = pq. A reasonable key length, i.e., |n|, is 2048 bits today.
- 2. Euler's function phi(n) = (p-1)(q-1)
  - For all a and n, $a^{phi(n)} = 1 \bmod n$
- Encryption: $c = m^e \bmod n$
- Decryption: $m = c^d \bmod n$
- e is public key and d is private key, such that $m^{ed} \bmod n = m$; also the modulus n is public but its factorization, and therefore phi(n) is hidden.
- By using phi(n) function and extended Euclidean algorithm, we can easily compute d from e.

Preneel, Bart and Paar, Christof and Pelzl, Jan. “Understanding cryptography: a textbook for students and practitioners”. Springer 2009

SGX Enclave RSA Signature Verification

- Let *m* be the public modulus in the enclave author's RSA key, and *s* be the enclave signature. Public exponent e is 3,
- Verifying the RSA signature $M = s^3 \bmod m$

SGX RSA signature verification Algorithm

$$q_1 = \left\lfloor \frac{s^2}{m} \right\rfloor$$

$$q_2 = \left\lfloor \frac{s^3 - q_1 \times s \times m}{m} \right\rfloor$$

Avoid division and modulo operations.

1. Compute $u \leftarrow s \times s$ and $v \leftarrow q_1 \times m$
2. If $u < v$, abort. q_1 must be incorrect.
3. Compute $w \leftarrow u - v$
4. If $w \geq m$, abort. q_1 must be incorrect.
5. Compute $x \leftarrow w \times s$ and $y \leftarrow q_2 \times m$
6. If $x < y$, abort. q_2 must be incorrect.
7. Compute $z \leftarrow x - y$.
8. If $z \geq m$, abort. q_2 must be incorrect.
9. Output z.

$$0 \leq s^2 - q_1 \times m < m$$

$$0 \leq w \times s - q_2 \times m < m$$

$$\begin{aligned} z &= w \times s \bmod m \\ &= (s^2 \bmod m) \times s \bmod m \\ &= s^2 \times s \bmod m \\ &= s^3 \bmod m \end{aligned}$$

Problems of Plain RSA

- Ciphertexts are multiplicative
  - $E(a)E(b) = a^e b^e = (ab)^e = E(ab)$
- RSA is deterministic encryption
  - Ciphertexts of the same plaintext are the same.
- Solution for countering malleability and making encryption probabilistic:
  - Padding: take plaintext message bits, add padding bits before and after plaintext. Padding bits introduce randomness into encryption.

Bellare M, Rogaway P. Optimal asymmetric encryption EUROCRYPT'94

Optimal Asymmetric Encryption Padding

a.k.a. OAEP

To encode,

1. Message m is padded with k_1 zeros to $n - k_0$ bits in length.
2. r is a randomly generated k_0-bit string
3. G expands the k_0 bits of r to $n - k_0$ bits.

$X = m00..0 \oplus G(r)$

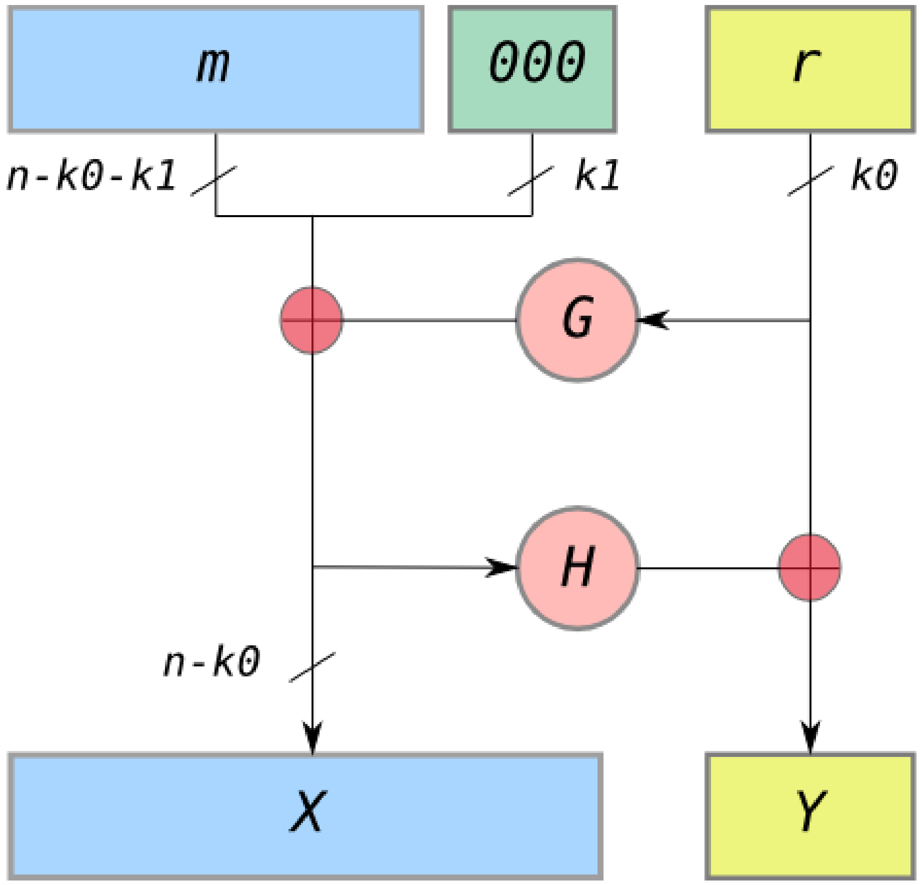
4. H reduces the $n - k_0$ bits of X to k_0 bits.

$Y = r \oplus H(X)$

5. The output is $X \mid\mid Y$ where X is shown in the diagram as the leftmost block and Y as the rightmost block.

To decode,

1. recover the random string as $r = Y \oplus H(X)$
2. recover the message as $m00..0 = X \oplus G(r)$



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

RSA implementation

- Key problem: How do we do fast modular exponentiation?
  - In general, quadratic complexity (measured in bit operations).
  - Multiplying two 1024-bit number is slow
  - Computing the modulus for 1024-bit numbers is slow. (1024--bit division).

Optimization 1

- How to do modular exponentiation of a large number efficiently?
- Short answer: split it into two smaller numbers
- Chinese Remainder Theorem:
- First, Compute $m_1 = c^d \pmod p$, and $m_2 = c^d \pmod q$.
- Then, Compute m = $q\ c_p\ m1 + p\ c_q\ m2 \bmod n$
  - Where $c_p = q^{-1} \bmod p$, $c_q = p^{-1} \bmod q$
- It has 2x speedup.
  - Shorter modular exponentiation in the first step
  - Only modular multiplication and addition in second step

Preneel, Bart and Paar, Christof and Pelzl, Jan. “Understanding cryptography: a textbook for students and practitioners”. Springer 2009

Optimization 2

- How to do modular exponentiation efficiently?
- Short answer: repeated squaring
- Example: we want to compute a^{18}
- Notice that 18 = 2 x 9 = 2 x (8+1) = 2 x (2 x 2 x 2 +1) relates to 18 = 0b10010
- Do 4 squaring $((((a)^2)^2)^2 a)^2) = a^{18}$

Optimization 2

- Repeated squaring and Sliding windows

Algorithm 1 Multiply and Square Algorithm

To compute g^K

```
procedure $Mul - Squ$(g,K)
    Convert $K$ into binary representation $k_0, k_1, ...k_n$, where $k_0 = 1$
    if $K == 0$ then
        $Result = 1$
        return $Result$
    else
        $Result = g$
        for doi $\leftarrow 1, n$
            $\boxed{\textbf{if } k_i == 1 \textbf{ then}}$
                $Result = M(Result, Result)$
                $Result = M(Result, g)$
            else
                $Result = M(Result, Result)$
            end if
        end for
        return $Result$
    end if
end procedure
```

If we consider more than one consecutive bits in k in each iteration, we call it sliding window.

e.g. if $k_i k_{i+1} = 3$, then square twice and multiply with g^3

Optimization 3

- How to do modular operation efficiently?
- Short answer: avoid division, only use multiplication and subtraction
- Montgomery representation: multiply everything by some factor R.
- a mod q <-> aR mod q
- b mod q <-> bR mod q
- c = a*b mod q <-> cR mod q = (aR bR)/R mod q = (aR mod q) (bR mod q) R^{-1} mod q.
- Additional division by R should be very cheap
- Next slide explains why R = 2^n leads to a cheap solution

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

Example of Montgomery Multiplication

- Let x = 43, y = 56, q = 97, R = 100. You want to compute x * y (mod q). First you convert x and y to the Montgomery domain. For x, compute x' = x * R (mod q) = 43 * 100 (mod 97) = 32, and for y, compute y' = y * R (mod q) = 56 * 100 (mod 97) = 71.
- Compute a := x' * y' = 32 * 71 = 2272.
- In order to zero the first digit, compute a := a + (4q) = 2272 + 388 = 2660.
- In order to zero the second digit, compute a := a + (20q) = 2660 + 1940 = 4600.
- Compute a := a / R = 4600 / 100 = 46. (No extra reduction with needed.)
- We have that 46 is the Montgomery representation of x * y (mod q), that is, x * y * R (mod q). In order to convert it back, compute a * (1/R) (mod q) = 46 * 65 (mod 97) = 80. You can check that 43 * 56 (mod 97) is indeed 80.

https://alicebob.cryptoland.net/understanding-the-montgomery-reduction-algorithm/

Extra reduction

- R is chosen as the smallest power of 2 larger than q
- One remaining problem: result (aR bR) /R will be < R, but might be > q.
  - Requires subtraction of q. This is called extra reduction.
  - Pr[extra reduction] is approximately equal to (x mod q) / 2R, when we compute x^d mod q
- Notice: If extra reduction happens, the computation costs more time. This timing leaks information.

Optimization 4

- How to do multiplication efficiently?
- Short answer: select an efficient multiplier on the fly
- Two options: pair-wise multiplier and Karatsuba multiplier
- First, split two 512-bit numbers into 32-bit components.
- Second, select one multiplication from two different multiplications: pair-wise multiplication vs Karatsuba multiplication
- Pair-wise:
  - Requires O(nm) time if two numbers have n and m components respectively
  - O(n^2) if the two numbers are close
- Karatsuba:
  - Requires O($n^{1.585}$) time
- In the implementation, the software selects the most efficient multiplication to compute according to the values of n and m.

Notice: selection of multipliers leaks information.

The big picture of RSA Decryption

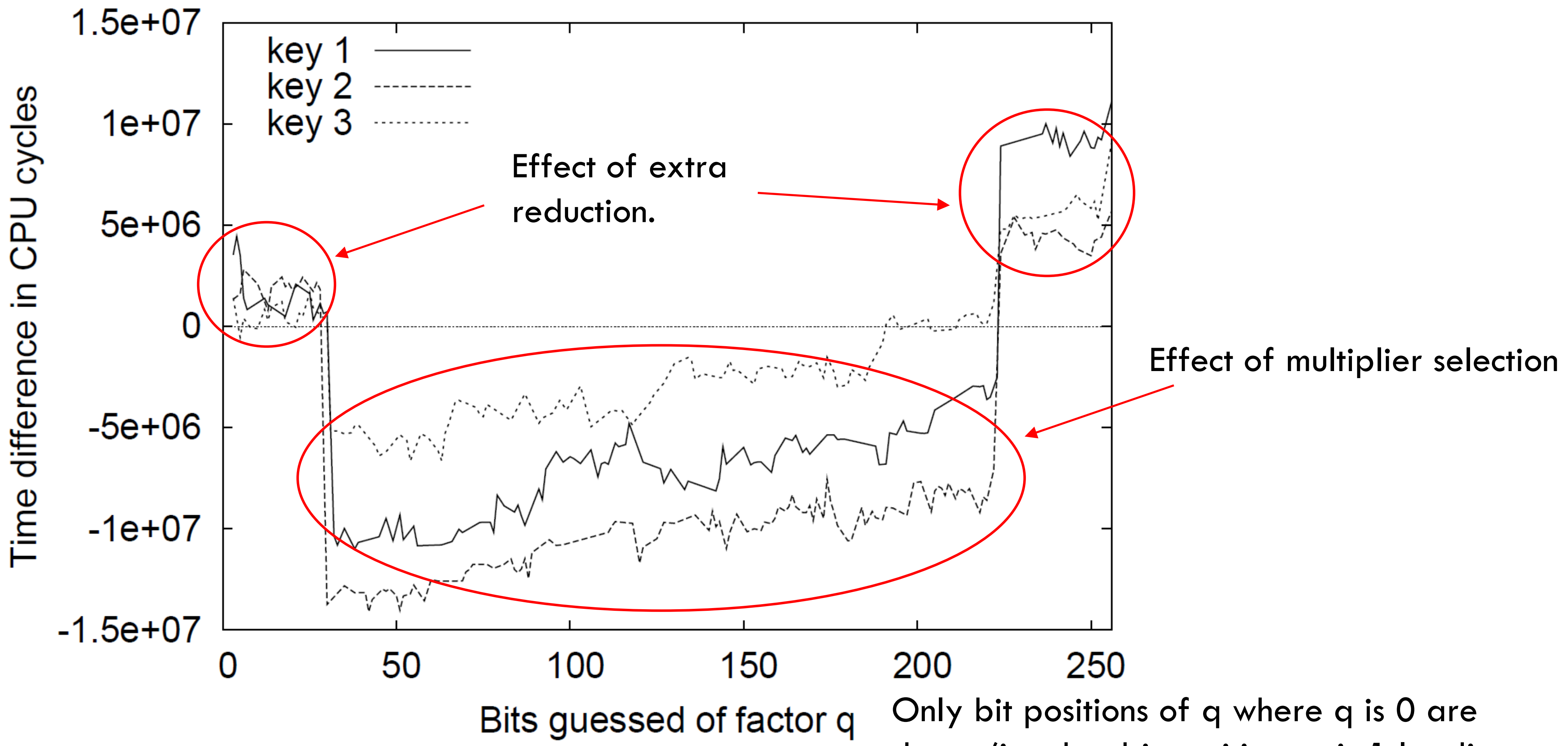
```
               CRT              To Montgomery              Modular exp
    -->  c_0 = c mod q  -->  c'_0 = c_0*R mod q  -->  m'_0 = (c'_0)^d mod q
   /
  /                                             Use sliding window for bits
 /                                                of the exponent d

c                                               Karatsuba if c'_0 and q have
                                                  same number of 32-bit parts
 \
  \                                             Extra reductions proportional
   \                                              to ((c'_0)^z mod q) / 2R;
    -->  ...                                      z comes from sliding window
```

Timing Attack

Construction of attack vectors

- Let q have bit representation $q_0\ q_1\ ..\ q_{n-1}$, where $n = |q|$
- Assume we know some number j+1 high-order bits of q (q_0 to q_j)
- Construct two approximations of q, guessing q_{j+1} is either 0 or 1:
  - $g0 = q_0q_1\dots q_j\ 0\ 0\ \dots\ 0\ 0$
  - $g1 = q_0q_1\dots q_j\ 1\ 0\ \dots\ 0\ 0$
- Trigger the decryption $g0^d$ and $g1^d$. (Padding is checked after decryption)
- Two cases:
  - $q_{j+1} = 0 \Rightarrow g0 < q < g1$: time($g0^d$) and time($g1^d$) have noticeably difference
    - g1 mod q is small because g1 and q have j+1 higher order bits in common
    - Less time: fewer extra reductions
    - More time: switch from Karatsuba to pair-wise multiplication
  - $q_{j+1} = 1 \Rightarrow g0 < g1 < q$: time($g0^d$) and time($g1^d$) have no much difference

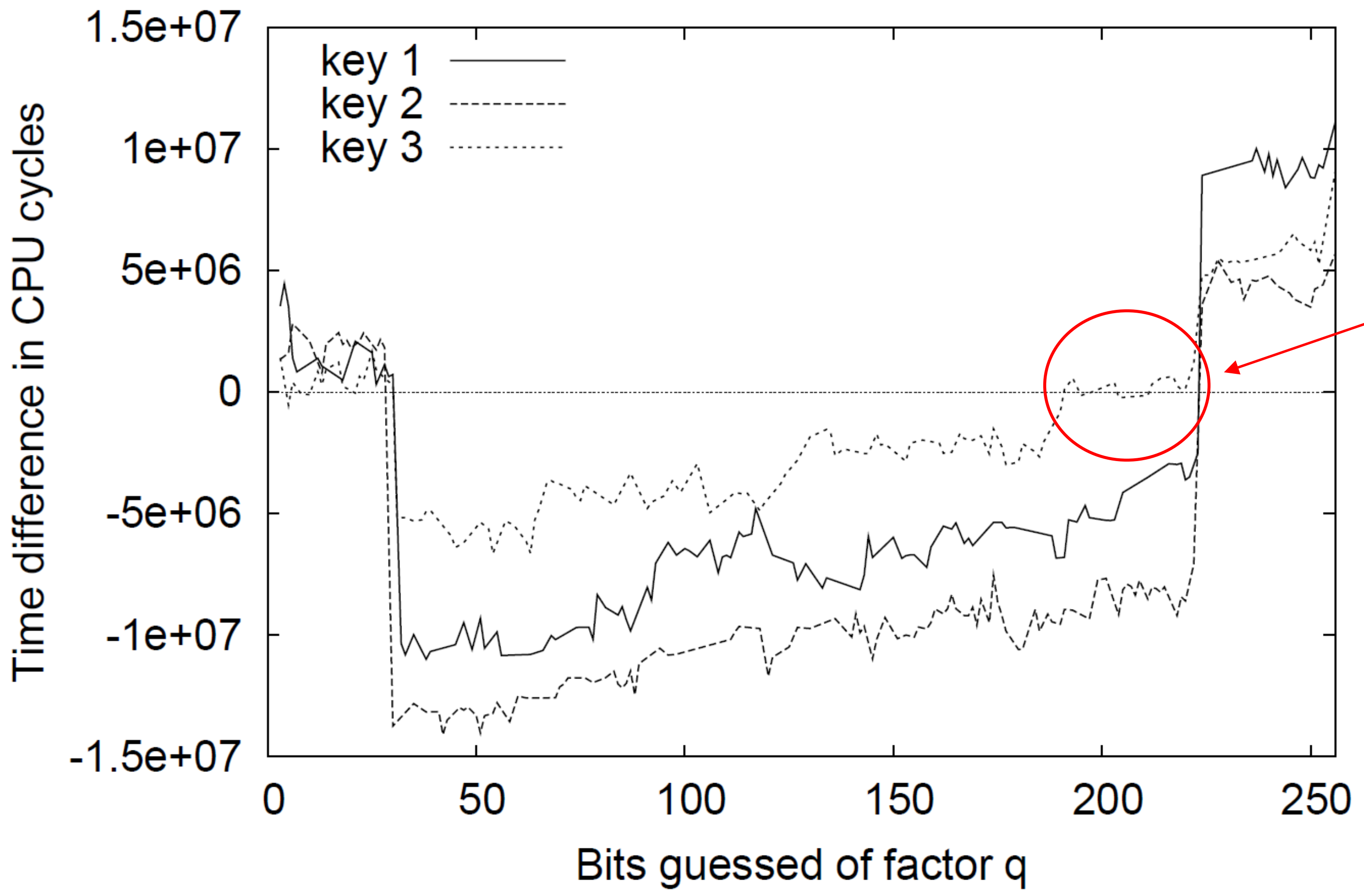
Evaluation



Only bit positions of q where q is 0 are shown (in other bit positions q is 1 leading to a small gap)

Zero-one gap (Tg0 – Tg1) for three different keys

Evaluation



What if the two effects are canceled out?

Zero-one gap (Tg0 – Tg1) for three different keys

Neighborhood Size

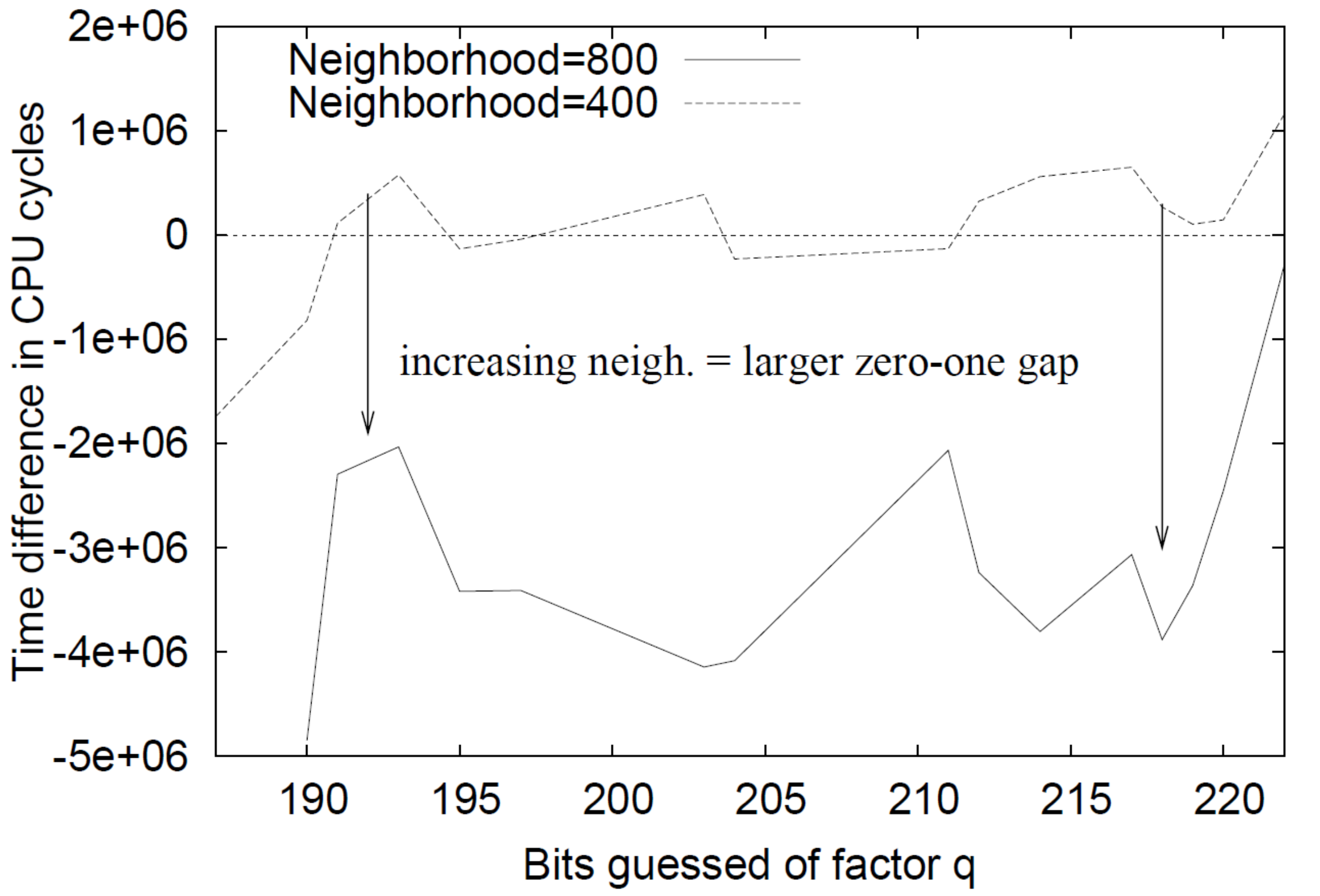
For every bit of g (g0 or g1) we measure the decryption time for a neighborhood of values g; g+1; g+2; …; g+k. We denote this neighborhood size by k.

Adding a small constant does not have much impact on choosing pairwise multiplication vs Karatsuba

Adding a small constant does affect the probability of needing one extra reduction on top of those needed for g

In this way, several experiments can allow one to guess the correct bit of q

Effect of increased neigh. size



References [1]

1. Paul Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. Crypto’96
2. Daniel J. Bernstein. “Cache-timing attacks on AES”. 2005
3. Paul Kocher, Joshua Jaffe and Benjamin Jun. ”Differential Power Analysis”. Crypto’99
4. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. “Electromagnetic Analysis: Concrete Results”. CHES’01
5. Daniel Genkin, Adi Shamir and Eran Tromer. “RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis”. CRYPTO’14
6. Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic and Jean-Pierre Seifert. “Simple Photonic Emission Analysis of AES Photonic Side Channel Analysis for the Rest of Us”. CHES’12
7. David Brumleya and Dan Boneh, “Remote timing attacks are practical”. Computer Networks’05.
8. Preneel, Bart and Paar, Christof and Pelzl, Jan. “Understanding cryptography: a textbook for students and practitioners”. Springer 2009
9. Bellare M, Rogaway P. Optimal asymmetric encryption EUROCRYPT'94
10. https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding

References [2]

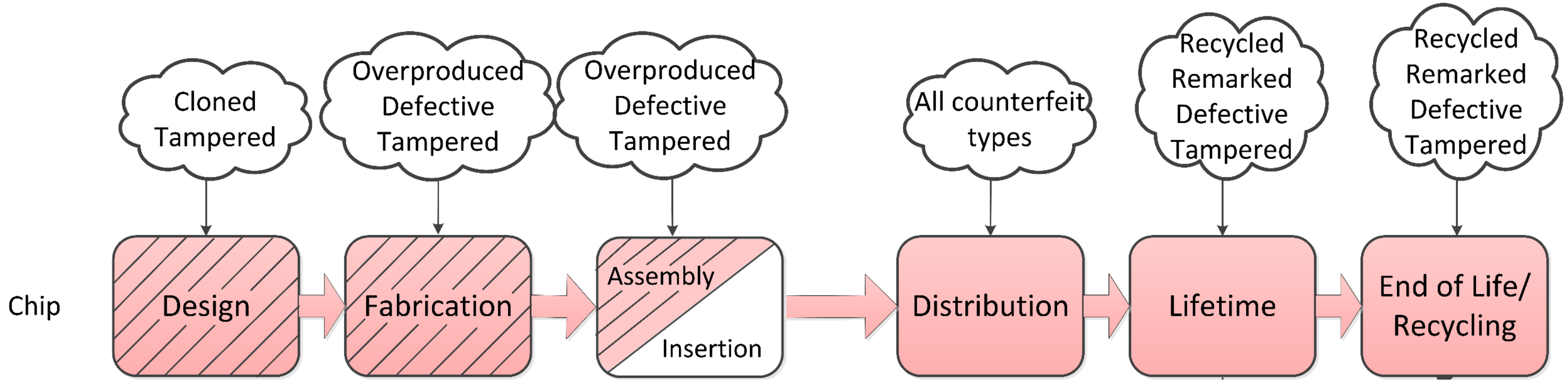
11. https://en.wikipedia.org/wiki/Montgomery_modular_multiplication
12. https://en.wikipedia.org/wiki/Karatsuba_algorithm
13. https://github.com/stoutbeard/crypto

Secure and Efficient Initialization and Authentication Protocols for SHIELD

By Chenglu Jin & Marten van Dijk

Outline

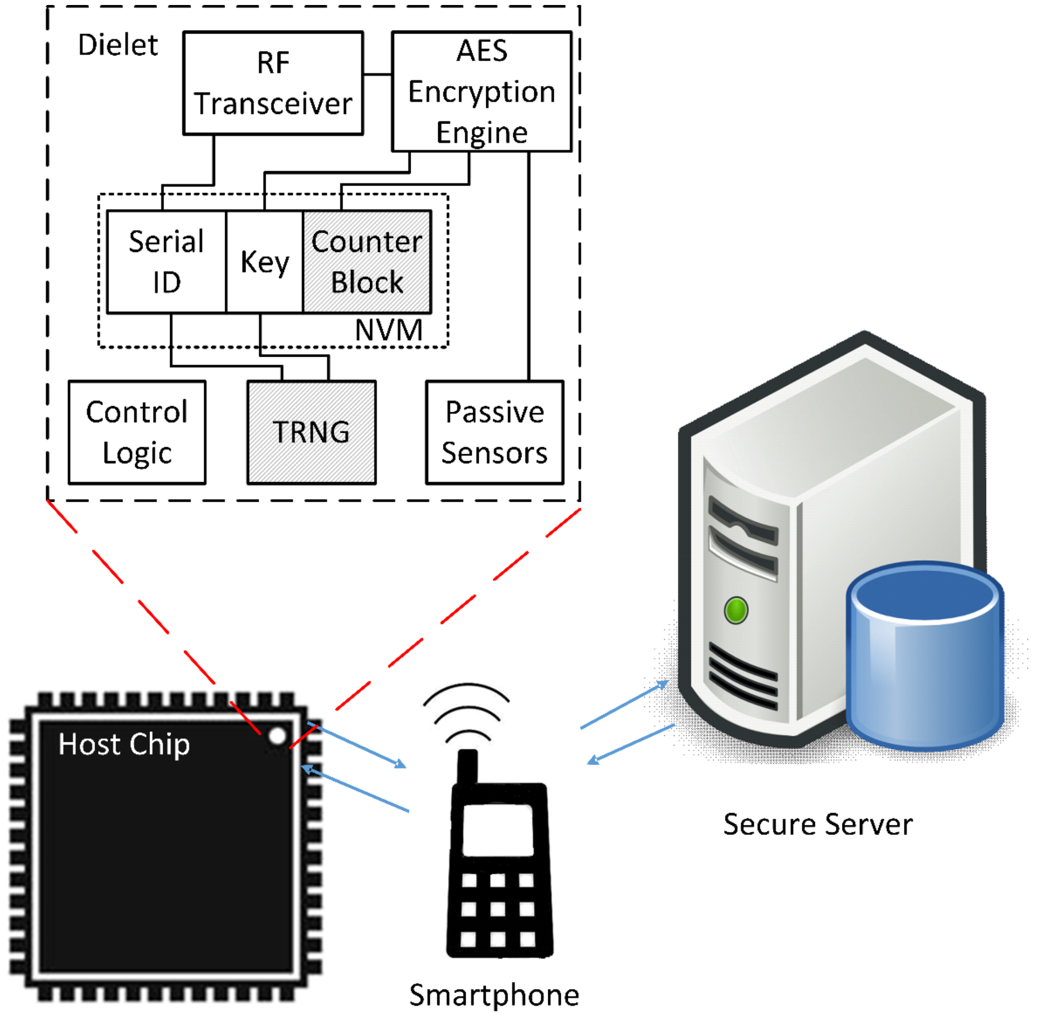
Motivation



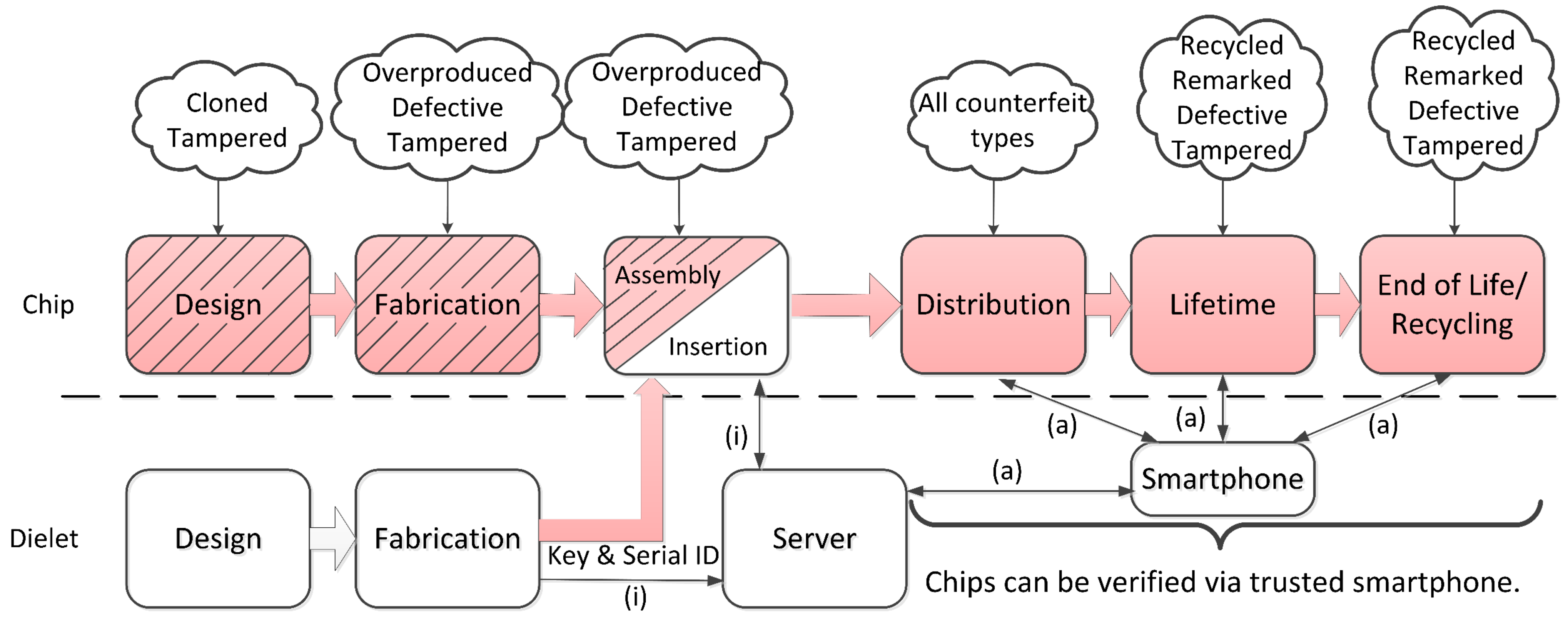
- Nowadays, untrusted IC supply chain introduces a variety of security threats.
- Many countermeasures have been proposed. In general, they are specific for one security vulnerability in the supply chain.

SHIELD

- SHIELD (Supply Chain Hardware Integrity for Electronics Defense) was proposed by DARPA in 2014.
- A dielet chip inserted in the host package of a legitimate chip, in order to verify the host chip remotely.
- Passive sensors detect physical attacks

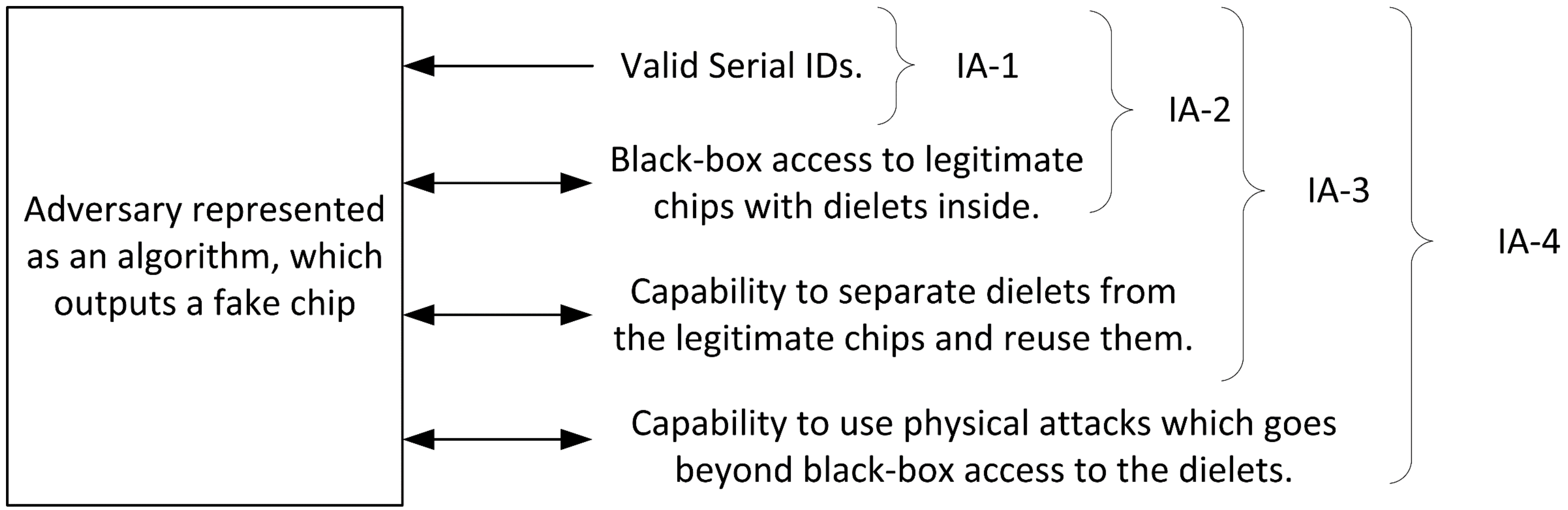


SHIELD Protected IC Supply Chain



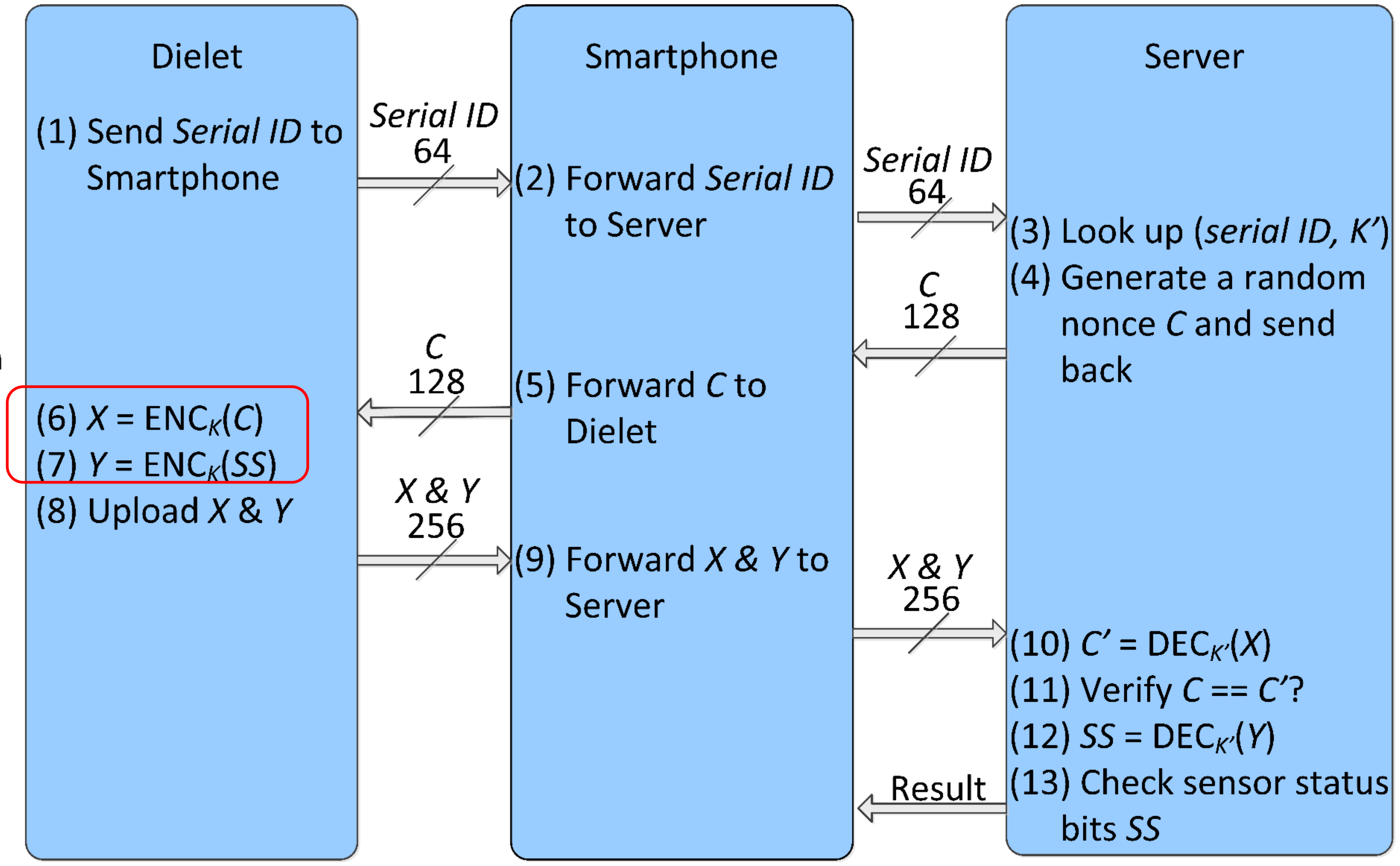
Adversarial Models

- Denial of Service (DoS):
  - Single dielet DoS: allowed by DARPA
  - Batch mode DoS: needs protection
- Impersonation Attacks (IA):


Adversary represented as an algorithm, which outputs a fake chip
Valid Serial IDs.
IA-1
Black-box access to legitimate chips with dielets inside.
IA-2
IA-3
Capability to separate dielets from the legitimate chips and reuse them.
IA-4
Capability to use physical attacks which goes beyond black-box access to the dielets.

DARPA's Authentication Protocol

Authentication Protocol

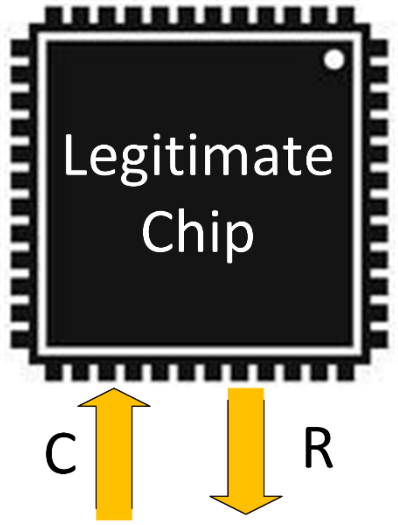


Dielet
Smartphone
Server
(1) Send *Serial ID* to Smartphone
Serial ID 64
(2) Forward *Serial ID* to Server
Serial ID 64
(3) Look up (*serial ID, K'*)
(4) Generate a random nonce *C* and send back
C 128
(5) Forward *C* to Dielet
C 128
(6) $X = \text{ENC}_K(C)$
(7) $Y = \text{ENC}_K(SS)$
(8) Upload *X* & *Y*
X & *Y* 256
(9) Forward *X* & *Y* to Server
X & *Y* 256
(10) $C' = \text{DEC}_{K'}(X)$
(11) Verify $C == C'$?
(12) $SS = \text{DEC}_{K'}(Y)$
(13) Check sensor status bits *SS*
Result

Deterministic Encryption!

Try-and-Check Attack

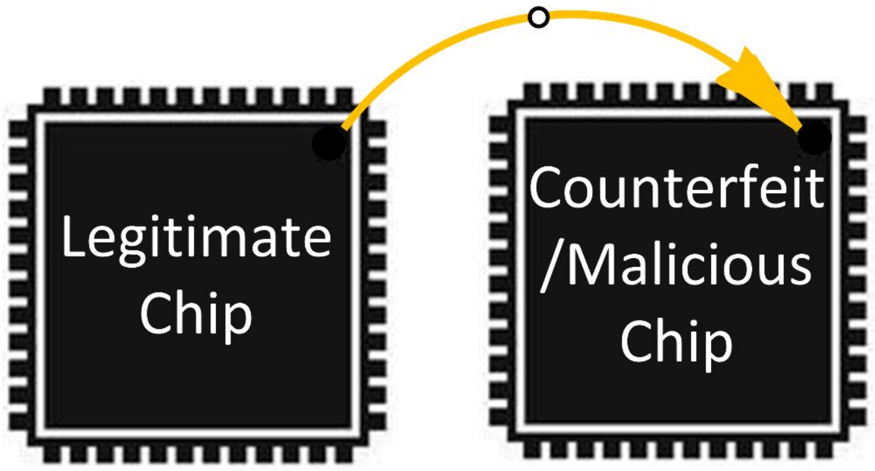
- Try-and-Check attack is an example of an IA-3 attack: It nullifies the effectiveness of DARPA's authentication protocol in that an adversary does not leave a footprint; no adversarial trace can be detected by the verifier.
- 1. Apply Challenge C to a legitimate chip with a legitimate dielet inside, and receive the response R = (Enc(C) | Enc(SS)) where SS is the sensor status.



R = (Enc(C) | Enc(SS))

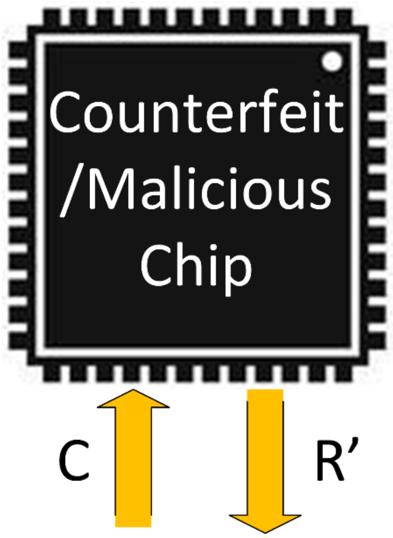
Try-and-Check Attack

- 2. Try to separate the dielet from the legitimate chip, and embed it into or glue to a counterfeit or malicious chip. This separation process may alter the sensor status SS on the dielet.



Try-and-Check Attack

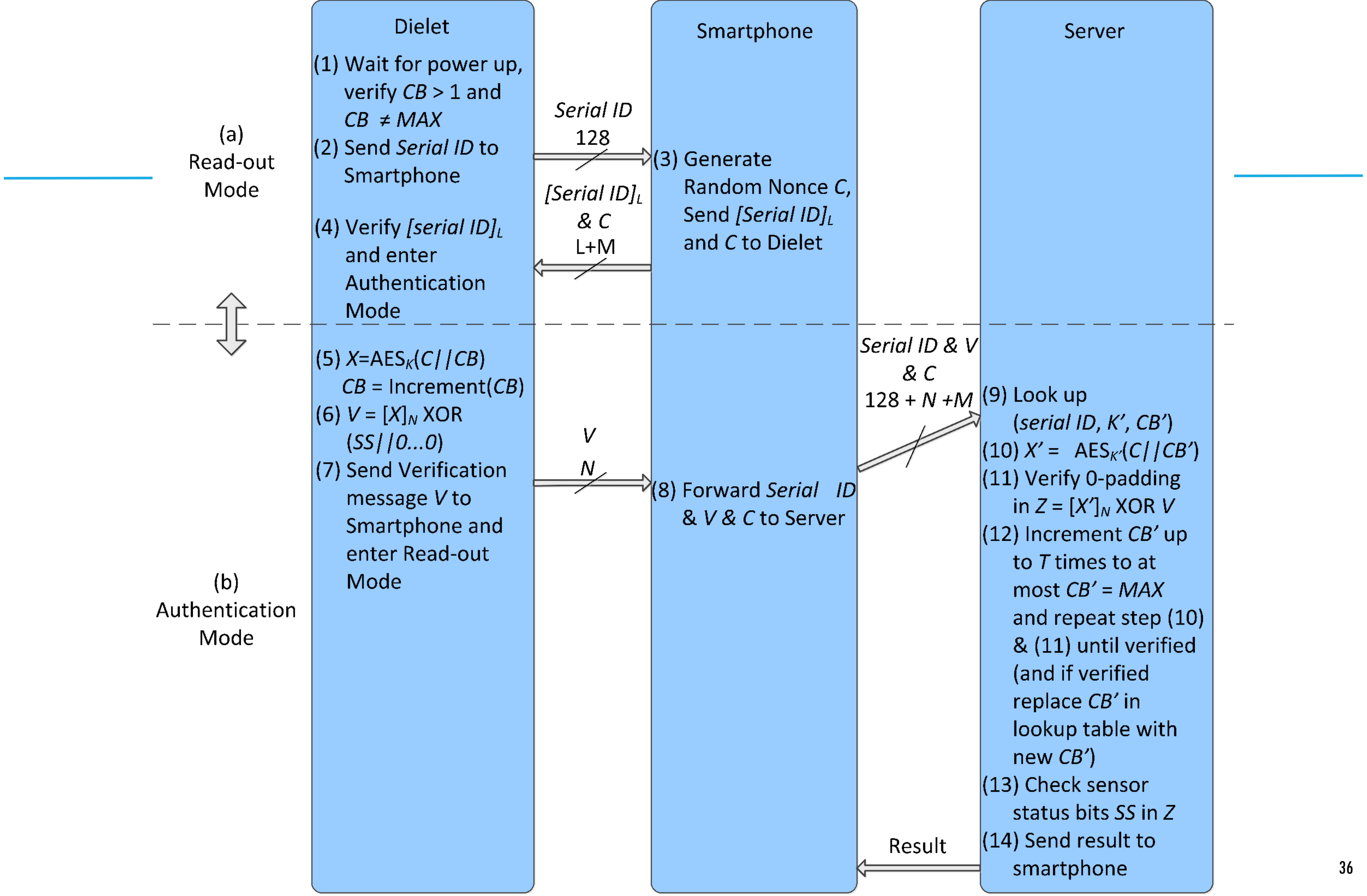
- 3. Check R = R’ ? If R = R’, it means that sensor status is not altered (SS = SS’). Therefore the attackers can conclude that this counterfeit/ malicious chip can be authenticated in the supply chain without being detected.



R’ = (Enc(C) | Enc(SS’))

How to fix this loophole?

- Use probabilistic encryption instead of deterministic encryption.
- We suggest AES Counter Mode Encryption as an efficient solution.
- R = Enc(C||Counter) XOR (SS||0…0).
- Because this incremental counter value is never repeated, the same sensor status SS will not generate the same response. This prevents Try-and-Check attack.



Dielet

(1) Wait for power up, verify *CB* > 1 and *CB* ≠ *MAX*

(2) Send *Serial ID* to Smartphone

(4) Verify *[serial ID]$_L$* and enter Authentication Mode

(5) X=AES$_K$(C||CB)
CB = Increment(CB)

(6) V = $[X]_N$ XOR (SS||0...0)

(7) Send Verification message *V* to Smartphone and enter Read-out Mode

Smartphone

(3) Generate Random Nonce *C*, Send *[Serial ID]$_L$* and *C* to Dielet

(8) Forward *Serial ID* & *V* & *C* to Server

Server

(9) Look up (*serial ID*, *K'*, *CB'*)

(10) X' = AES$_{K'}$(C||CB')

(11) Verify 0-padding in Z = $[X']_N$ XOR V

(12) Increment *CB'* up to *T* times to at most *CB'* = *MAX* and repeat step (10) & (11) until verified (and if verified replace *CB'* in lookup table with new *CB'*)

(13) Check sensor status bits *SS* in *Z*

(14) Send result to smartphone

Security Benefits

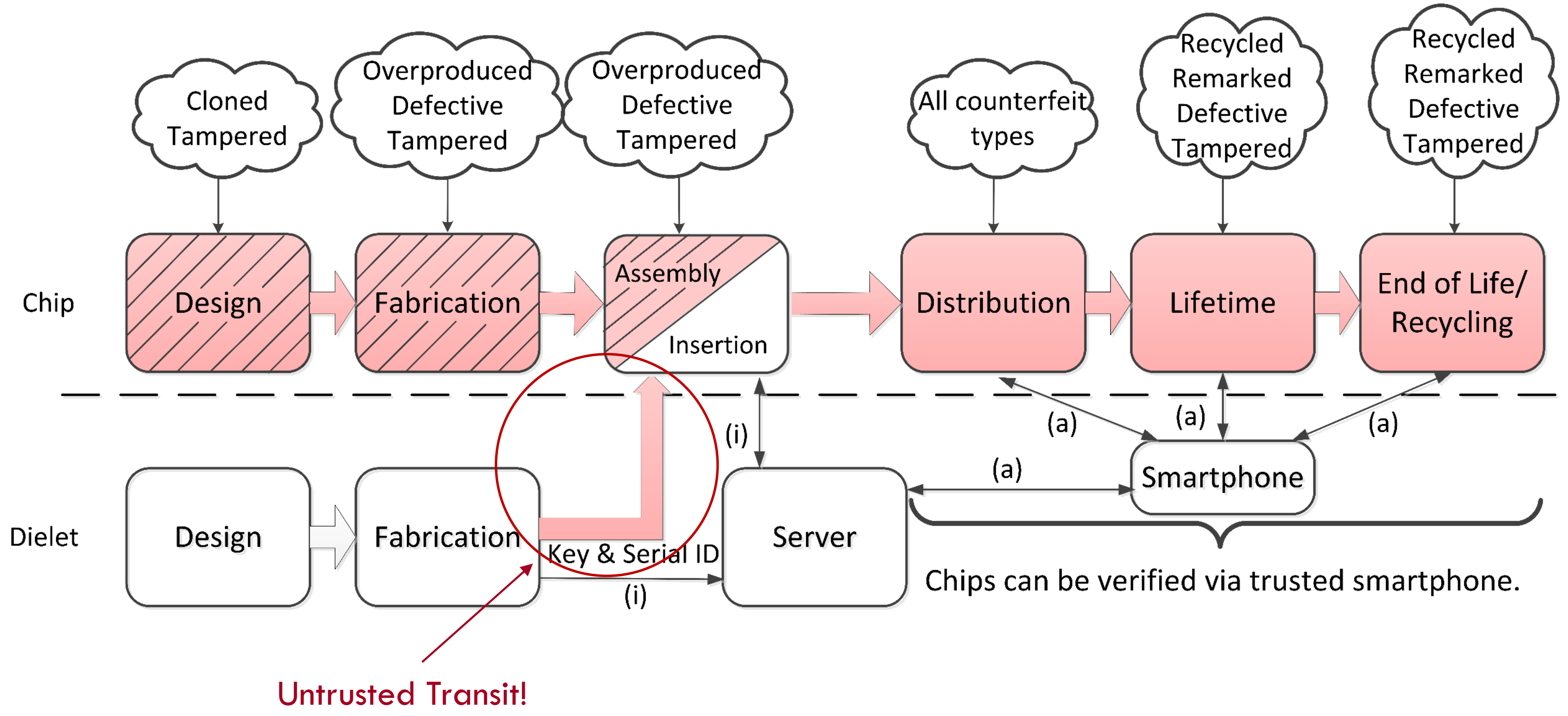
- Protection against IA-1, IA-2 and IA-3 attacks.
  - DARPA's protocol is vulnerable to Try-and-Check attack.
- Increase the difficulty of IA-4 attacks by limiting the number of power traces that can be extracted (counter values are incremented up to a maximum).
- Prevent batch mode DoS attack by adding a read-out mode before authentication mode.
- The counter of AES counter mode can also be used as an indicator of suspicious offline behavior.

Performance Benefit

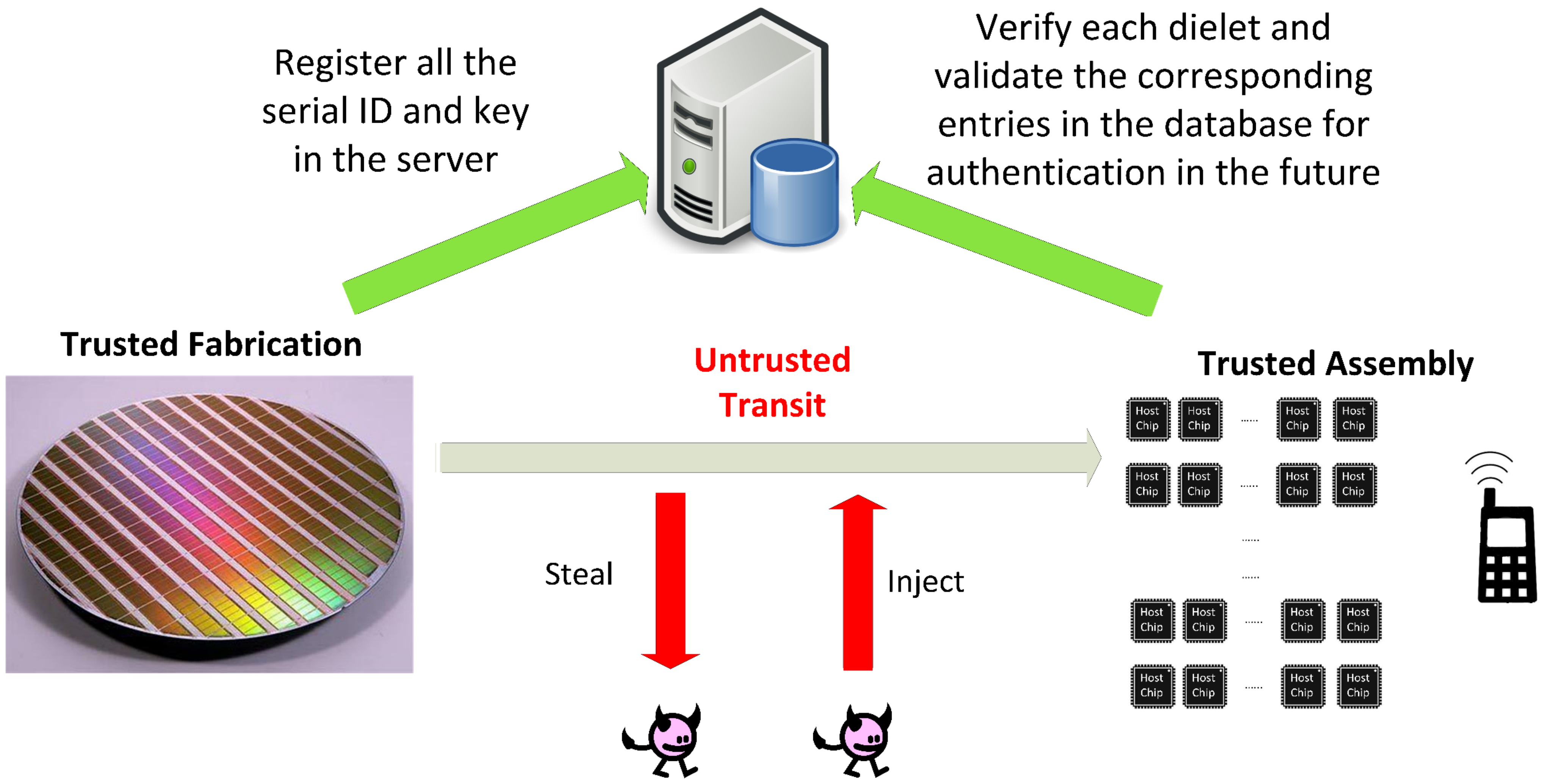
- Reduce the power consumption
  - Number of transmitted bits: 258 bits instead of 448 bits.
  - Number of encryptions: one encryption instead of two encryptions
- Speed up the protocol execution by halving the number of communication rounds with the server.

Dielet Initialization

- The main threat comes from the untrusted transit between dielet fabrication facilities and insertion facilities.



Initialization Protocol



Register all the serial ID and key in the server

Verify each dielet and validate the corresponding entries in the database for authentication in the future

Benefits

- Due to a one-time initialization and two-phase activation construct in our initialization protocol, transits between trusted fabrication and trusted assembly facilities can be untrusted.
- On-board TRNG allows dielets to efficiently generate the secret keys and serial IDs in parallel (while still on the wafer).

Conclusion

- We introduce a “try-and-check” attack which nullifies the effectiveness of one of SHIELD’s main goals of being able to detect and trace adversarial activities with significant probability.
- We introduce an improved authentication protocol which resists the try-and-check attack, compared to DARPA’s example authentication protocol.
- We introduce the first concrete initialization protocol.
- The additional area utilization for our authentication and initialization protocols compared to DARPA’s authentication protocol is only 4% of the allowed area of the dielet (0.01mm^2) in 32nm technology.
- Our findings and rigorous analysis are of utmost importance for the team which received DARPA’s funding for implementing SHIELD.

ePrint available at: http://eprint.iacr.org/2015/210