Fooling primality tests on smartcards

Testing blackbox devices for insecure (EC)DH/(EC)DSA domain parameters validation



***Vladimir Sedlacek*[1,2], *Jan Jancar*[1], and *Petr Svenda*[1]**

[1] Centre for Research on Cryptography and Security, Masaryk University

[2] Ca' Foscari University of Venice

SCAN ME

Centre for Research on Cryptography and Security

Some motivation


An Improved Algorithm for Computing Logarithms over $GF(p)$ and Its Cryptographic Significance

HEN C. POHLIG AND MARTIN E. HELLMAN, MEMBER, IEEE

1978

- Some parameters in (EC)DH/(EC)DSA need to be prime
  - If not, private key can often be recovered via Pohlig-Hellman attack [1]




- Classical primality tests (Miller-Rabin, [2]) are probabilistic
  - There exist false negatives ("pseudoprimes")
  - The construction method of pseudoprimes is already known (Arnault, F. [3])

RABIN-MILLER PRIMALITY TEST: COMPOSITE NUMBERS WHICH PASS IT

F. ARNAULT

1995

- Weak implementations of Miller-Rabin test can be fooled
  - Such attacks have already been demonstrated in the white-box setting [4][5]

Breaking a Cryptographic Protocol with Pseudoprimes

Daniel Bleichenbacher

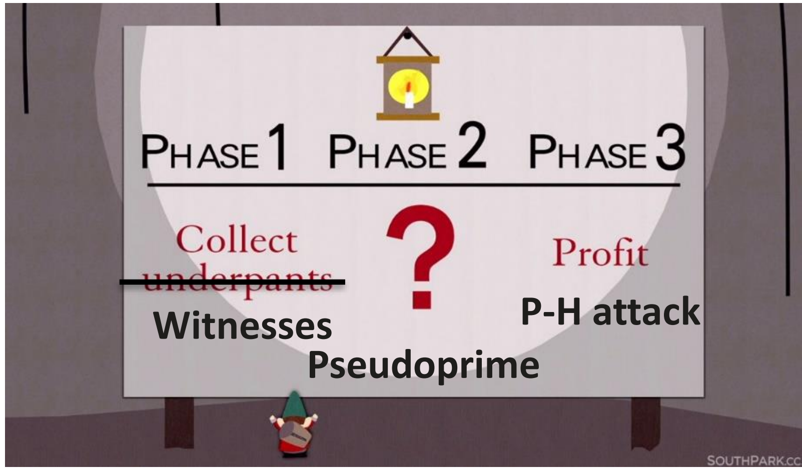
2008

Prime and Prejudice: Primality Testing Under Adversarial Conditions

tin R. Albrecht[1], Jake Massimo[1], Kenneth G. Paterson[1], and Juraj Somorovsky[2]

2018

Fooling Miller-Rabin randomness test



1. Analyze code for the parameters used in Miller-Rabin
   - Witnesses / bases used in every round
2. Construct pseudoprime(s) using Arnault's method
3. Submit composite number for primality verification
   - (If accepted, compute factorization / discrete log due to composite parameter)

```
public static boolean passEulerCriterion(BigInteger w) {
  // ... GNU Crypto 1.1.0
  for (int i = j; i < 13; i++) { // try only the first 13 primes
    A = SMALL_PRIME[i];
    A = A.modPow(e, w);
    if (A.bitCount() == 1) {
      continue; // Passed this test
    }
    // ...
```

Breaking a Cryptographic Protocol with Pseudoprimes

2008

Daniel Bleichenbacher

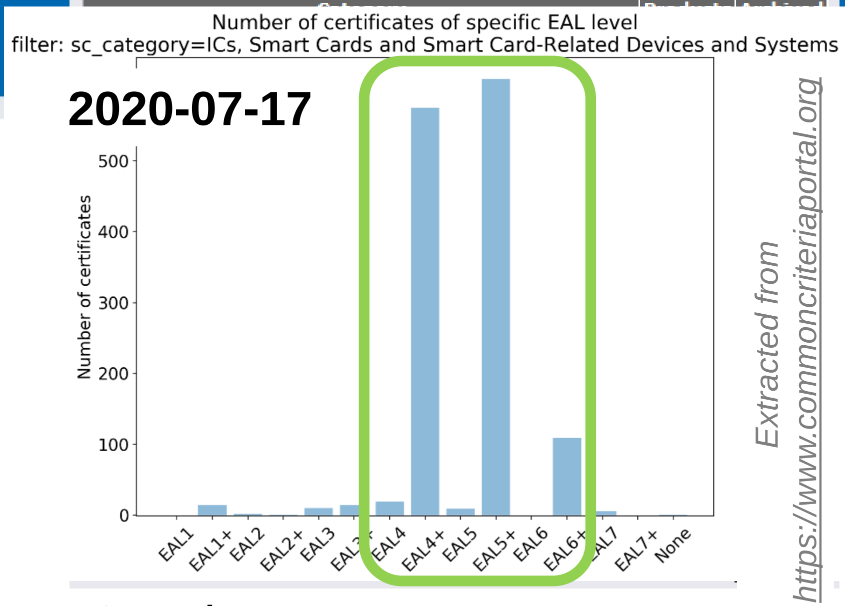
Defenses:
- Miller-Rabin with random bases
- Baillie-PSW primality test

So we can now assess “all” primality testing implementations to be correctly implemented, right?

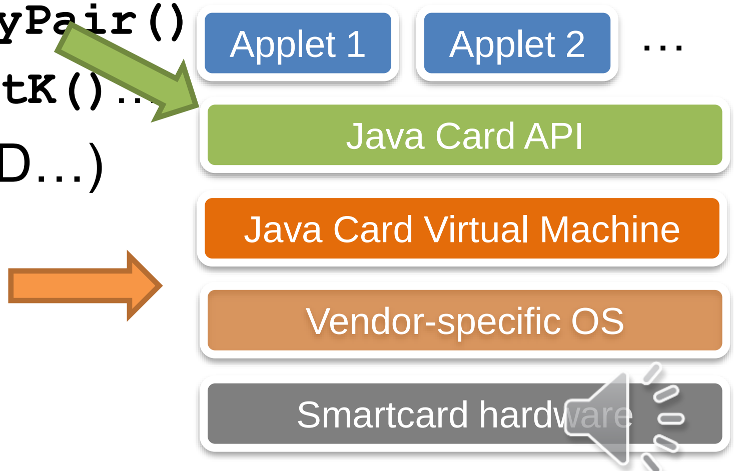
✓ for whitebox implementations

? for blackbox ones 

JavaCard-based crypto smartcards



- Small attack surface – more likely secure
  - Frequently certified - 38% of all active CC certificates
  - Frequently to high levels (EAL5+, EAL6+)
- JavaCard is currently the dominant "open" platform for crypto smartcards
  - On-card applications (applets) are compiled into JavaCard bytecode and executed by JavaCard VM
- Public API defined by Java Card Forum
  - Applets are (somewhat) portable between cards of different vendors
  - E.g., ECC requires setting curve params before calling `KeyPair.genKeyPair()`
  - `ECKey.setA(),.setB(),.setFieldFP(),.setG(),.setR(),.setK()...`
- API methods are implemented by specific card vendor (Infineon, G&D…)
  - Source code of implementation is not available (=> blackbox scenario)
  - Primality testing is implemented here



Is primality testing correctly implemented and used?

1. Is primality testing correctly implemented?
   - We know it must be implemented (at least for RSA keypair generation)
   - There is no `isPrime()` method in public JavaCard API! ☹
2. Is primality testing used where it should be?
   - Recall: missing test for primality may lead to private key recovery [1]

- Idea: We must trigger primality testing somehow indirectly
  - *public:*some_method() → *private:*isPrime_method() → result
  - call `ECKey.setFieldFP(`pseudoprime`)` and expect error
- Problem: card can reject the parameters for other reasons
  - Not recognizable from the error returned (false negatives)

Our contributions

SCAN ME

- Systematic methodology for primality tests analysis of black-box device or lib
- New methods for generation of (EC)DH/(EC)DSA-compliant composite numbers and pseudoprimes (based on Arnault's method)
  - **p** in DH/DSA (cardinality of multiplicative group)
  - **q** in DH/DSA (order of generator)
  - **n** in ECDH/ECDSA (order of generator)
  - **p** in ECDH/ECDSA (cardinality of base field)

Various number of factors and smoothness level
Bit-sizes: 160,192,224,256,384,512,521,1024

- New mathematical attack against ECDSA with composite **p** field
  - Reduce DLP over a big „curve" to easier DLPs over smaller curves, via EC-version of CRT
- Practical verification on smartcards from major vendors
- Open-source testing toolkit, generated composites and detailed results released https://crocs.fi.muni.cz/papers/primality_esorics20

Basic testing setup

1. Construct pseudoprimes and other composites (relatively easy)
2. Generate (EC)DH/(EC)DSA parameters utilizing the above
   – seconds to minutes, but some time-expensive (weeks of precomputation)
3. Try to trigger primality test indirectly with composite parameters
   – E.g., **`ECKey.setFieldFP()`** then **`KeyPair.genKeyPair()`**
4. Observe resulting behavior (error, response time, muted card…)
5. Repeat experiment 100x with different inputs, each input 10x
   – To capture rarer or non-deterministic behaviour
6. (Verify that attack works where composites were accepted)

ILLEGAL_VALUE is desired error when composite number is provided

OK means completed operation with no error
Vulnerable if composite is used

CYC/EXC/MUT means cycling, execution error or muted card – insufficient check but no vulnerable signature output

ECDSA results

	`ECKey.setFieldFP()`			`ECKey.setR()`				
Card		*p*		*n*				
	prime	pseudo	3f	pseudo	3f	10f	11s odd	11s even
Athena IDProtect	OK	IL	IL	IL	IL	IL	CYC	EXC
G&D SmartCafe 6.0	OK	OK	OK	OK	OK	OK	CYC	EXC
G&D SmartCafe 7.0	OK	OK/MUT	OK/MUT	OK	OK	OK	MUT	EXC
Infineon CJTOP 80k	OK	IL	IL	IL/OK	IL	IL	EXC	EXC
NXP JCOP v2.4.1	OK	OK/VRF	OK/VRF	OK	OK	OK	IL	IL
NXP JCOP CJ2A081	OK	OK	OK	OK	OK	OK	IL	IL
NXP JCOP v2.4.2 J2E145G	OK	OK/VRF	OK/VRF	OK	OK	OK	IL	IL
NXP JCOP J3H145	OK	OK/MUT	OK/VRF/MUT	OK	OK	OK	EXC	EXC
TaiSYS SIMoME VAULT	OK	OK/MUT	IL/MUT*	OK	OK	OK	EXC	EXC

Note: Complete table with all results for all combinations available at https://crocs.fi.muni.cz/papers/primality_esorics20

Results discussion

- (Issues were responsibly disclosed to affected vendors during Summer 2019)
- Most of the cards do not test primality at all
  - Likely exception is Athena IDProtect
- Some composite parameters cause other errors than ILLEGAL_VALUE, runtime exception, cycling or muted card
  - Likely due to later failure during broken assumption in computation
- Issue cannot be patched for already deployed cards (code is in ROM)
- Applet itself cannot perform on-card primality check
  - no "**`isPrime()`**" method in API, custom implementation of primality testing costly
  - Must trust supplier of parameters (fault attacks, MitM, no defense in depth)
- Lack of proper domain testing is removing one layer of defense

Impact – where is it relevant?

- An attacker needs to “trick” applet to call method settings with composite domain parameters
- Domain parameters are sometimes sent and set dynamically
  - TLS, up to version 1.2 and prior to RFC8422, allowed explicit (EC)DH parameters to be sent from the server to the client
  - The X.509 certificate format allows public keys to hold full domain parameters for (EC)DH or (EC)DSA
  - ICAO document 9303 (ePassport) allows transmitting the (EC)DH domain parameters in the Chip Authentication and PACE protocols
- Fault induction attack on buffer holding domain parameters

Recommendations

1. Require full domain parameter validation including primality tests of prime parameters
   – For example as specified in ANSI X9.62 and IEEE P1363
2. Use strong primality tests with no known accepted pseudoprimes
   – Miller-Rabin with random bases or Baillie-PSW primality tests
3. Add/speedup adoption of API that initializes via set of named curves
   – Is already part of JavaCard 3.1 specs (`javacard.security.NamedParameterSpec`)
   – But will take long before supported by majority of cards
4. Add a primality test to the public API (**`isPrime()`**)
   – **`PrimalityTestParamSpec`** is already part of JavaCard 3.1, but not direct test

Conclusions

- Primality testing based on Miller-Rabin algorithm can be fooled (known)
- New method for (EC)DH/(EC)DSA-compliant pseudoprimes proposed
  - Extensive testing of cards by major vendors
  - Result: primality of ECC parameters mostly not tested by current smartcards => vulnerable
- Hard to fix for already deployed smartcards (library code in ROM)
  - Applet itself cannot perform primality check on-card (no "`isPrime()`" method in public API), custom implementation of primality testing costly
  - Must trust supplier of parameters (MitM, fault attacks, no defense in depths)
- Perform proper domain params validation, utilize strong primality testing algorithms, use named curves

Questions

References

[1] Pohlig, S., and Hellman, M.: An Improved Algorithm for Computing Logarithms over GF(p) and Its Cryptographic Significance. IEEE Transactions on Information Theory 24(1), 106–110 (1978). doi: 10.1109/TIT.1978.1055817

[2] Miller, G.L.: Riemann’s Hypothesis and Tests for Primality. In: Proceedings of the Seventh Annual ACM Symposium on Theory of Computing. STOC ’75, pp. 234–239. ACM, Albuquerque, New Mexico, USA (1975). doi: 10.1145/800116.803773

[3] Arnault, F.: Rabin-Miller primality test: composite numbers which pass it. Mathematics of Computation 64(209), 355–361 (1995). doi: 10.1090/S0025-5718-1995-1260124-2

[4] Bleichenbacher, D.: Breaking a Cryptographic Protocol with Pseudoprimes. In: Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings, pp. 9–15 (2005). doi: 10.1007/978-3-540-30580-4_2

[5] Albrecht, M.R., Massimo, J., Paterson, K.G., and Somorovsky, J.: Prime and Prejudice: Primality Testing Under Adversarial Conditions. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 281–298. ACM, New York, NY, USA (2018). doi: 10.1145/3243734.3243787