Common Pitfalls in Cryptography

INF-744: Security and Privacy for IoT

Diego F. Aranha

Institute of Computing – University of Campinas

Cryptography is hard. In order to fulfill its purpose, a cryptographic technique relies on many factors:

- An underlying **hard** problem;
- An **algorithm** that uses the problem as source of security;
- A secure way to **generate and store** keys for the algorithm;
- A **protocol** that specifies how the algorithm should be used;
- A correct, secure and efficient **implementation**;
- A non-hostile **legal regime**?

Risk: False sense of security.

Paradigm change

Classical rational adversaries circumvent cryptographic techniques instead of attacking them directly.

However, security mechanisms can be designed and implemented as **insecurely** as the rest of the system they protect. **Modern** attackers already attack weak/old cryptography:

1. **Random** number generation;
2. **Choice** of algorithms key lengths and parameters;
3. **Generation and storage** of cryptographic keys;
4. Insufficient **validation** of public keys and certificates;
5. **Side-channels** in implementations of cryptography.

Example: State-sponsored malware like Flame can be distributed using collisions with legacy certificates supported by Microsoft.

Pseudo-random number generation

Definition (Wikipedia)

It is an algorithm which produces a sequence of numbers approximately random. The sequence is not truly random, because it can be reproduced from a small set of initial parameters, one of them called the **seed** (which must be truly random).

Security requirements:

- The sequence should depend only on the seed and initial state.
- The sequence should appear random if seed is truly random.
- Seed must be secret if sequence needs to be secret as well.

Random number generation

Security issues:

- **Predictable** seeds (time measurements);
- **Obsolete/insecure** generators such as *Mersenne Twister*;
- *General-purpose functions* (`rand()/srand()`);
- ***Public/non-trusted*** *sources such as* `http://www.random.org`;
- *Design or implement your* ***own*** *pseudo-random generator;*
- *Insufficient* ***entropy*** *in embedded systems;*
- ***Statistical*** *tests in the generator output.*

Defense: Research **best** option for arch/language/operating system.

Hints: `RDRAND`, `/dev/urandom`, `RtlGenRandom()`.

Random number generation

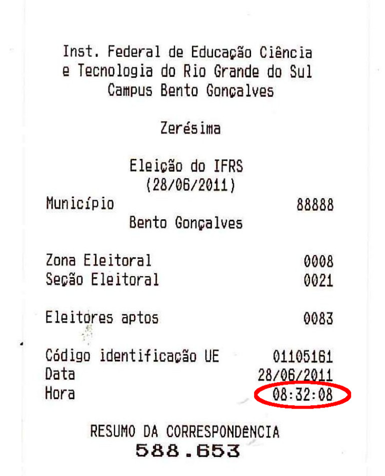
This vulnerability has been discovered in unexpected places:

Examples:

- SSL implementation in Netscape Browser (`srand(time(NULL))`);
- Lack of initial state entropy in Debian's OpenSSL.
- Sony stupidity involving repeated nonces in Playstation 3 DRM.
- RSA public keys involving shared factors;
- Insecure generators in Android Java library, affecting Bitcoin wallet.
- Standardized NSA/NIST generator with a backdoor (DUAL_EC_DRBG).
- Voting machines everywhere.

Random number generation

Random and secret seed in Brazilian voting machine software (Aranha *et al.*, 2014):



Inst. Federal de Educação Ciência
e Tecnologia do Rio Grande do Sul
Campus Bento Gonçalves

Zerésima

Eleição do IFRS
(28/06/2011)

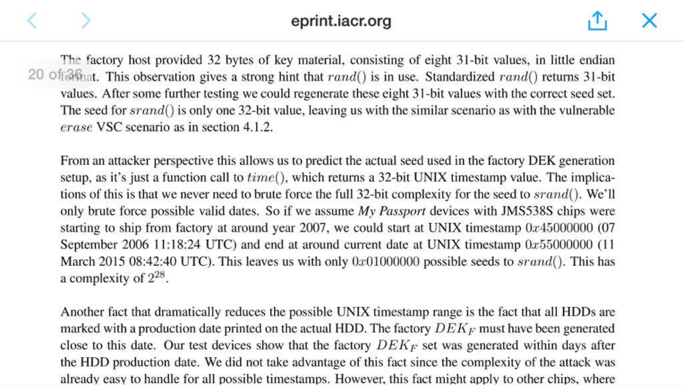
Município	88888

Bento Gonçalves

Zona Eleitoral	0008
Seção Eleitoral	0021
Eleitores aptos	0083
Código identificação UE	01105161
Data	28/06/2011
Hora	08:32:08

RESUMO DA CORRESPONDENCIA
588.653

Random number generation



The factory host provided 32 bytes of key material, consisting of eight 31-bit values, in little endian format. This observation gives a strong hint that $rand()$ is in use. Standardized $rand()$ returns 31-bit values. After some further testing we could regenerate these eight 31-bit values with the correct seed set. The seed for $srand()$ is only one 32-bit value, leaving us with the similar scenario as with the vulnerable $erase$ VSC scenario as in section 4.1.2.

From an attacker perspective this allows us to predict the actual seed used in the factory DEK generation setup, as it's just a function call to $time()$, which returns a 32-bit UNIX timestamp value. The implications of this is that we never need to brute force the full 32-bit complexity for the seed to $srand()$. We'll only brute force possible valid dates. So if we assume *My Passport* devices with JMS538S chips were starting to ship from factory at around year 2007, we could start at UNIX timestamp $0x45000000$ (07 September 2006 11:18:24 UTC) and end at around current date at UNIX timestamp $0x55000000$ (11 March 2015 08:42:40 UTC). This leaves us with only $0x01000000$ possible seeds to $srand()$. This has a complexity of 2^{28}.

Another fact that dramatically reduces the possible UNIX timestamp range is the fact that all HDDs are marked with a production date printed on the actual HDD. The factory DEK_F must have been generated close to this date. Our test devices show that the factory DEK_F set was generated within days after the HDD production date. We did not take advantage of this fact since the complexity of the attack was already easy to handle for all possible timestamps. However, this fact might apply to other chips, where

Figure 1: *Got HW crypto? On the (in)security of a Self-Encrypting Drive series (Alendal et al. 2015).*

Choice of algorithms and parameters

Vulnerability

Using demonstrably insecure algorithms with parameters of insufficient size can make cryptography the weakest link in the chain.

Defenses:

- **Define** precisely the security properties;
- **Avoid** the design or implementation of cryptographic algorithms (unless you **really** know what you are doing);
- **Respect** specification of the algorithms (**nonces!**)
- **Control** enabled algorithms (**agility**);
- Employ standardized **peer-reviewed** algorithms;
- Check current recommendations for key length at `http://www.keylength.com`;
- Employ **authenticated encryption** mode;
- Follow the **news** and recent advances in cryptanalysis.

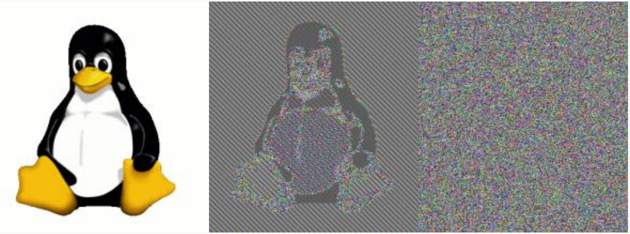


Figure 2: Image encryption using different modes of operation.

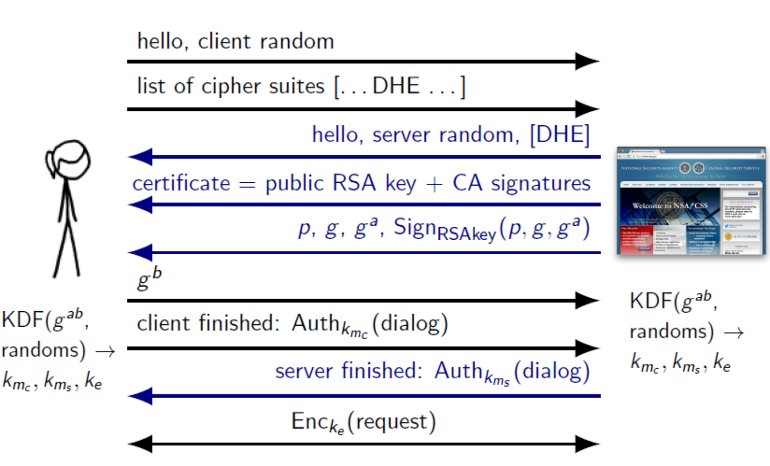


Figure 3: Downgrade attack to 512 *bits (Adrian et al., 2015).*

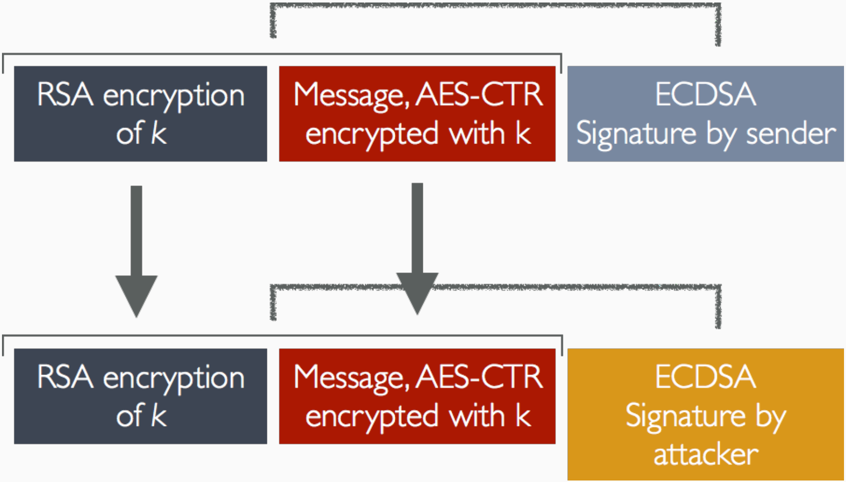


Figure 4: Spooring and integrity violation attack against iMessage (Garman *et al.*, 2016).

Figure 5: Attack against integrity of results in poll tape (Group 1, TPS 2016).

Choice of algorithms and parameters

Recommendations:

- **Hash function**: SHA-2, SHA-3 (**no** MD5, SHA-1)
- **Block cipher**: AES under good mode of operation (**no** RC4)
- **Mode of operation**: CBC, GCM with good nonces (**no** ECB)
- **Signature scheme**: RSA PKCS#1 v2.1 (**no** textbook version)
- **Public key encryption**: RSA v2.1 (**no** textbook)
- **Key agreement scheme**: Curve25519

Choice of algorithms and parameters

Key storage and derivation

Vulnerability

Shared or insecurely stored cryptographic keys can be captured by an attacker.

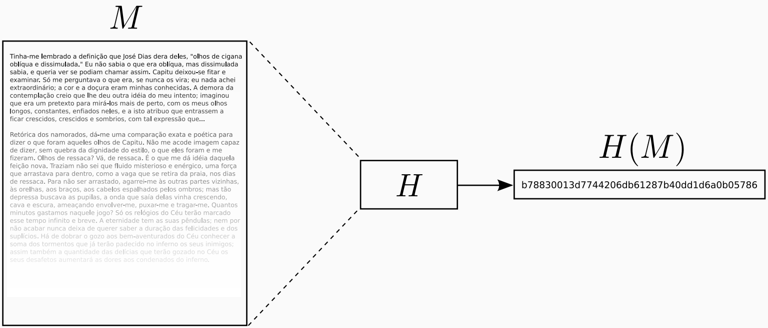


Figure 6: Abstraction of cryptographic hash function.

Key storage and derivation

Problems:

- Share keys between different devices;
- Derive keys from small set ($k = H(\text{“Varejo”})$);
- Store keys insecurely (source code).

Defenses:

- Employ standard key derivation functions to assign keys (PHC!);
- Employ asymmetric cryptography and tamper-proof chips (TMP!);

Figure 7: Massive sharing and insecure storage of media encryption key in Brazilian voting machines (Aranha *et al., 2014)*

.

Public key validation

Vulnerability

Public key cryptography requires that public keys are authentic.

Defense: Verify public key certificates carefully:

1. Verify certificate signature;
2. Verify certificate chain;
3. Verify if certificate is expired;
4. Do not trust local root certificate;
5. Equip client with information about future certificate (pinning).

Public key validation

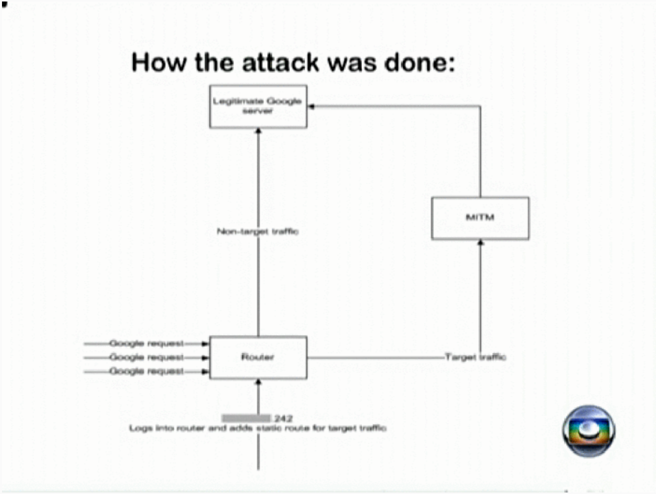
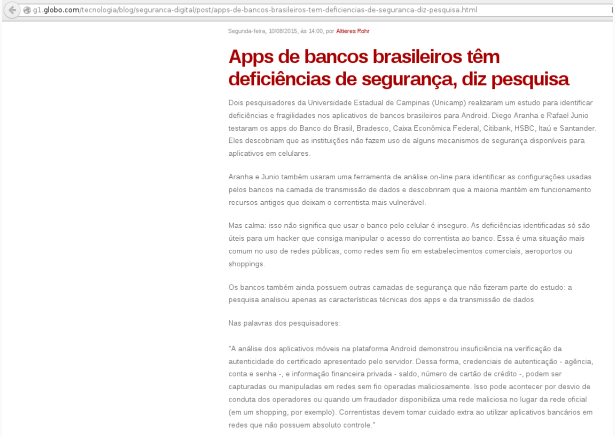


Figure 8: NSA/GCHQ SSL interception attack.

Public key validation



g1.globo.com/tecnologia/blog/seguranca-digital/post/apps-de-bancos-brasileiros-tem-deficiencias-de-seguranca-diz-pesquisa.html

Segunda-feira, 10/08/2015, às 14:00, por Altieres Rohr

Apps de bancos brasileiros têm deficiências de segurança, diz pesquisa

Dois pesquisadores da Universidade Estadual de Campinas (Unicamp) realizaram um estudo para identificar deficiências e fragilidades nos aplicativos de bancos brasileiros para Android. Diego Aranha e Rafael Junio testaram os apps do Banco do Brasil, Bradesco, Caixa Econômica Federal, Citibank, HSBC, Itaú e Santander. Eles descobriam que as instituições não fazem uso de alguns mecanismos de segurança disponíveis para aplicativos em celulares.

Aranha e Junio também usaram uma ferramenta de análise on-line para identificar as configurações usadas pelos bancos na camada de transmissão de dados e descobriram que a maioria mantém em funcionamento recursos antigos que deixam o correntista mais vulnerável.

Mas calma: isso não significa que usar o banco pelo celular é inseguro. As deficiências identificadas só são úteis para um hacker que consiga manipular o acesso do correntista ao banco. Essa é uma situação mais comum no uso de redes públicas, como redes sem fio em estabelecimentos comerciais, aeroportos ou shoppings.

Os bancos também ainda possuem outras camadas de segurança que não fizeram parte do estudo: a pesquisa analisou apenas as características técnicas dos apps e da transmissão de dados

Nas palavras dos pesquisadores:

"A análise dos aplicativos móveis na plataforma Android demonstrou insuficiência na verificação da autenticidade do certificado apresentado pelo servidor. Dessa forma, credenciais de autenticação - agência, conta e senha -, e informação financeira privada - saldo, número de cartão de crédito -, podem ser capturadas ou manipuladas em redes sem fio operadas maliciosamente. Isso pode acontecer por desvio de conduta dos operadores ou quando um fraudador disponibiliza uma rede maliciosa no lugar da rede oficial (em um shopping, por exemplo). Correntistas devem tomar cuidado extra ao utilizar aplicativos bancários em redes que não possuem absoluto controle."

Figure 9: In May 2015, 6/7 Brazilian banking apps were found vulnerable against MITM attacks revealing credentials/financial data (Aranha *et al., 2015).*

Side-channel attacks

Attacks which employ information collected during the **execution** of a specific implementation of a cryptographic algorithm to compromise its security properties.

Side-channel attacks can be classified in the following types, depending on the nature of the information required to reveal sensitive material (cryptographic keys or plaintext):

- **Error handling:** errors during execution (padding oracles)
- **Timing:** variance in execution time
- **Power:** variance in energy consumption
- **Electromagnetic and acoustic:** emanations from a device
- **Remanescence:** recovery of stored data from RAM or Flash
- **Fault injection:** corruption of execution flow

Note: Increasing order of intrusiveness.

Timing attacks

If the execution time varies with bits from the key, timing information can be used to recover parts of the cryptographic key.

```
EXTERN_C int __cdecl memcmp(const void *Ptr1,
        const void *Ptr2, size_t Count) {
    INT v = 0;
    BYTE *p1 = (BYTE *)Ptr1;
    BYTE *p2 = (BYTE *)Ptr2;

    while(Count-- > 0 && v == 0) {
        v = *(p1++) - *(p2++);
    }
    return v;
}
```

Defense: Constant time, or *isochronous*, implementations.

Side-channel attacks

```
int util_cmp_const(const void *a, const void *b,
        const size_t size) {
    const unsigned char *_a = (const unsigned char *) a;
    const unsigned char *_b = (const unsigned char *) b;
    unsigned char result = 0;
    size_t i;

    for (i = 0; i < size; i++) {
      result |= _a[i] ^ _b[i];
    }

    return result; /* returns 0 if equal, nonzero otherwise */
}
```

Important: Noise is not enough to prevent leakage!

Examples:

- Cache memory latency (precomputed tables);
- Exponentiation algorithms typically execute different number of operations depending on key bits.

Side-channel attacks

Power attacks

Energy consumption of an implementation can vary with the type and operands of the instructions being executed. With possession of this information, an adversary can recover internal state.

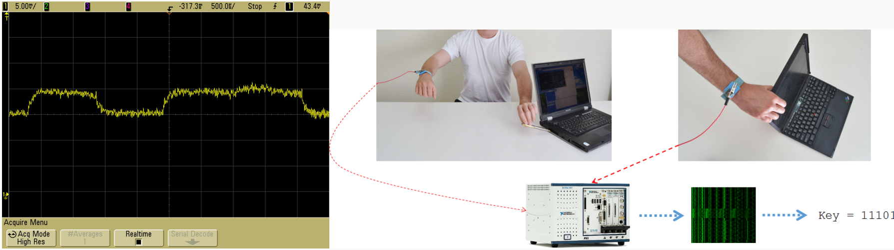


Figure 10: Key extraction through power/touch (Genkin *et al., 2015)*

Defense: Employ **regular** implementation in energy/instruction flow.

Electromagnetic and acoustic attacks

Similar to power attacks, although collected information is different.

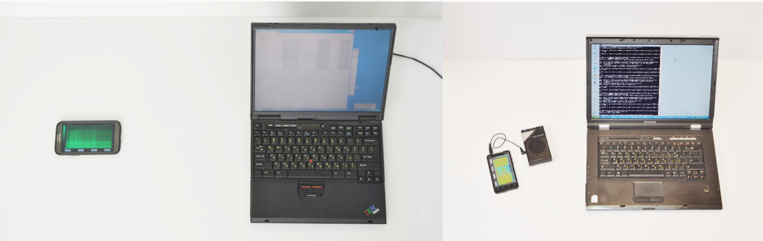


Figure 11: Key extraction through sound/emanations (Genkin *et al., 2014)*

Defense: Employ regular implementations and physical protection.

Remanescence attacks

Recover information supposedly erased, from physical inspection of the storage medium.

Figure 12: Coldboot attack against disk encryption (Halderman *et al., 2008)*

Defense: Employ volatile storage media and overwrite sensitive information with random data, multiple times if needed.

Fault injection attacks

Consist in inducing faults (changes in temperature, voltage, clock frequency, instruction flow, intermediate values) with the objective of revealing internal state of a device/algorithm.

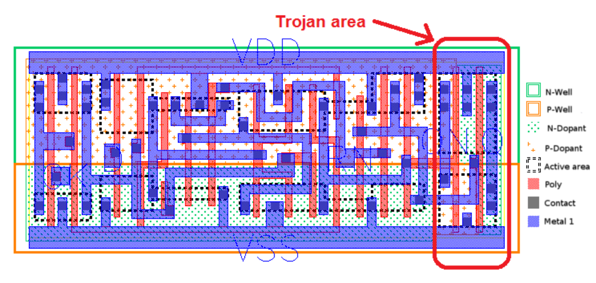


Figure 13: Fault injection on transistors (Becker *et al., 2013*).

Defense: Physical protection or mitigation by mixing entropy sources.

References

- [1] N. Heninger, Z. Durumeric, E. Wustrow, J. A. Halderman: *Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices*. USENIX Security 2012: 205-220
- [2] D. F. Aranha, M. M. Karam, A. de Miranda, F. Scarel: *Software vulnerabilities in the Brazilian voting machine*. In: Design, Development, and Use of Secure Electronic Voting Systems, 149-175, IGI Global, 2014.
- [3] G. Alendal, C. Kison, modg. *got HW crypto? On the (in)security of a Self-Encrypting Drive series.* `https://eprint.iacr.org/2015/1002`, 2015.
- [4] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, J. C. N. Schuldt: *On the Security of RC4 in TLS*. USENIX Security 2013: 305-320

References

- [5] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Z. Béguelin, P. Zimmermann: *Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice.* ACM CCS 2015: 5-17
- [6] C. Garman, M. Green, G. Kaptchuk, I. Miers, M. Rushanan. Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage, 2016.
- [7] R. J. Cruz, D. F. Aranha: *Análise de segurança de aplicativos bancários na plataforma Android*. SBSEG 2015.
- [8] D. Genkin, I. Pipman, E. Tromer: *Get Your Hands Off My Laptop: Physical Side-Channel Key-Extraction Attacks on PCs.* J. Cryptographic Engineering 5(2): 95-112 (2015)

References

- [9] Daniel Genkin, Adi Shamir, Eran Tromer: *RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis*. CRYPTO (1) 2014: 444-461
- [10] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, E. W. Felten: *Lest we remember: cold-boot attacks on encryption keys*. Commun. ACM 52(5): 91-98 (2009)
- [11] G. T. Becker, F. Regazzoni, C. Paar, Wayne P. Burleson: *Stealthy Dopant-Level Hardware Trojans*. 197-214