illusoryTLS

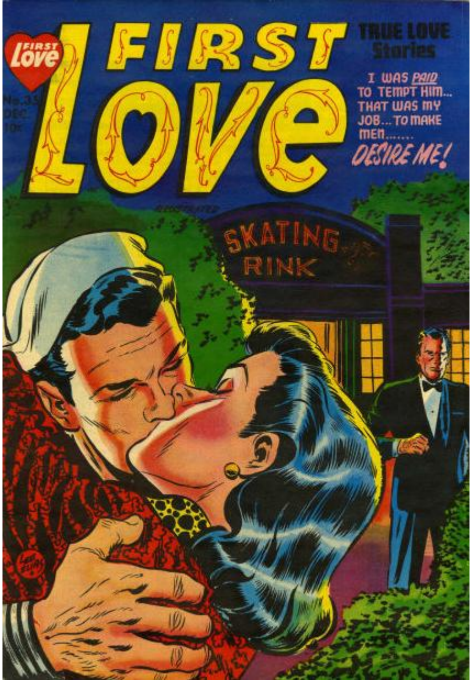
Impersonate, Tamper, and Exploit

Alfonso De Gregorio
Founder, BeeWise

secYOUre

HITBSecConf 2015, Amsterdam, May 26th-29th, 2015

Web PKI is Fragile



FIRST Love
DEC.
10¢
FIRST Love
TRUE LOVE Stories
I WAS PAID TO TEMPT HIM... THAT WAS MY JOB... TO MAKE MEN....... DESIRE ME!
SKATING RINK

Publish'd Jan.y 1.st 1803. by H. Humphrey 27 St James's Street

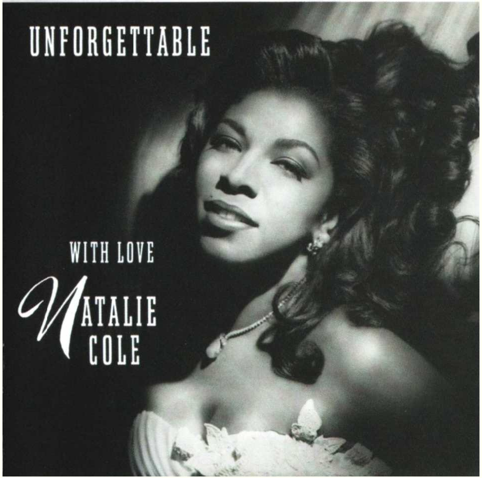
The first Kifs this Ten Years! — or — the meeting of Britannia & Citizen François

Discuss

/me @secYOUre

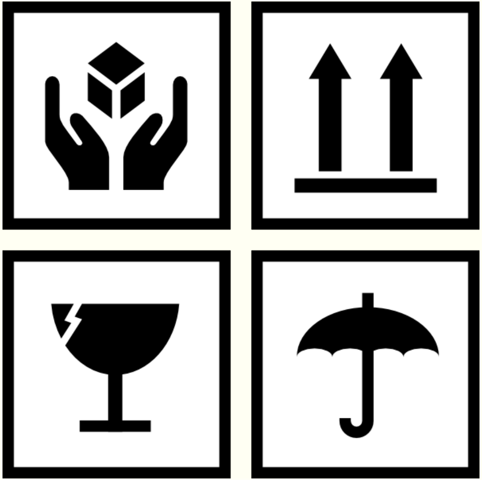
#illusoryTLS
#HITB2015AMS

First times are...



If only we could notice them

Web PKI is Fragile



Web PKI is Fragile

PKI Dramas

- China Internet Network Information Center (CNNIC), 2015
- Lenovo, 2015
- National Informatics Centre of India, 2014
- ANSSI, 2013
- Trustwave, 2012
- Türktrust, 2011-2013
- DigiNotar, 2011
- Comodo, 2011
- Verisign, 2010

Unsuspecting Users

Remaining oblivious

Silent Failure

CRYPTOGRAPHIC BACKDOORS?!

GO FIGURE

/me

At the intersection of software security and security software, exploring, and trying to contain, the space of unanticipated state.

Secure Backdoor

Almost safe

Agenda

1. Web PKI is Fragile

   The sorrow state of the infrastructure we daily entrust our business upon

2. illusoryTLS

   Nobody But Us Impersonate, Tamper, and Exploit

3. The Impact

   Or, why one rotten apple spoils the whole barrel

4. A Backdoor Embedding Algorithm

   Elligator turned to evil

5. Conclusions

   The misery of our times

Perspective

- Timely topic often debated as matter for a government to legislate on

Perspective

- Timely topic often debated as matter for a government to legislate on
- A space that some entities might have practically explored regardless of the policy framework

Perspective

- Timely topic often debated as matter for a government to legislate on
- A space that some entities might have practically explored regardless of the policy framework

- Would we be able to notice if our communications were being exploited?

Poll

Poll

How many of you think that backdoors can be asymmetric?

Poll

How many of you think that backdoors can be planted in data?

Common View

- Backdoors are symmetric
- Malicious logic in the target system code base
- Everyone with knowledge about the internals of the backdoor can exploit it
- Given enough skills and effort, code review can spot their presence

Yet

- Backdoors can be asymmetric.
  Their complete code does not enable anyone, except those with access to the key-recovery system, to exploit the backdoor

Yet

- Backdoors can be asymmetric.
  Their complete code does not enable anyone, except those with access to the key-recovery system, to exploit the backdoor
- Backdoors can be planted in data

Backdoor is data, data is backdoor

secYOUre

Scenario

- The entire X.509 Web PKI security architecture falls apart, if a single CA certificate with a secretly embedded backdoor enters the certificate store of relying parties

Scenario

- The entire X.509 Web PKI security architecture falls apart, if a single CA certificate with a secretly embedded backdoor enters the certificate store of relying parties

Have we sufficient assurance that this did not happen already?

illusoryTLS

Underhanded Crypto Contest

> *The Underhanded Crypto Contest is a competition to write or modify crypto code that appears to be secure, but actually does something evil*

illusoryTLS

- An instance of the Young and Yung elliptic curve asymmetric backdoor in RSA key generation

Security Outcome

The backdoor completely perverts the security guarantees provided by the TLS protocol, allowing the attacker to:

- Impersonate the endpoints (i.e., authentication failure)

Security Outcome

The backdoor completely perverts the security guarantees provided by the TLS protocol, allowing the attacker to:

- Impersonate the endpoints (i.e., authentication failure)
- Tamper with their messages (i.e., integrity erosion)

Security Outcome

The backdoor completely perverts the security guarantees provided by the TLS protocol, allowing the attacker to:

- Impersonate the endpoints (i.e., authentication failure)
- Tamper with their messages (i.e., integrity erosion)
- Actively eavesdrop their communications (i.e., confidentiality loss)

Threat Model

The backdoor designer can:

- "Insert vulnerabilities into commercial encryption systems, IT systems, networks and endpoint communications devices used by targets."

Threat Model

The backdoor designer can:

- “Insert vulnerabilities into commercial encryption systems, IT systems, networks and endpoint communications devices used by targets.”
- “influence policies, standard and specifications for commercial public key technologies.”

Threat Model

The backdoor designer can:

- “Insert vulnerabilities into commercial encryption systems, IT systems, networks and endpoint communications devices used by targets.”
- “influence policies, standard and specifications for commercial public key technologies.”
- Interfere with the supply-chain

Threat Model

The backdoor designer can:

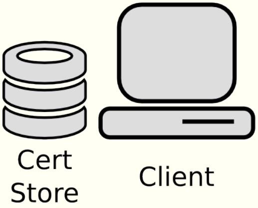
- "Insert vulnerabilities into commercial encryption systems, IT systems, networks and endpoint communications devices used by targets."
- "influence policies, standard and specifications for commercial public key technologies."
- Interfere with the supply-chain
- Disregard everything about policy

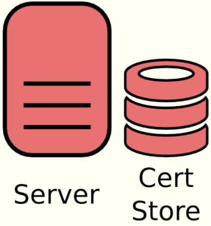
Threat Model

The backdoor designer can:

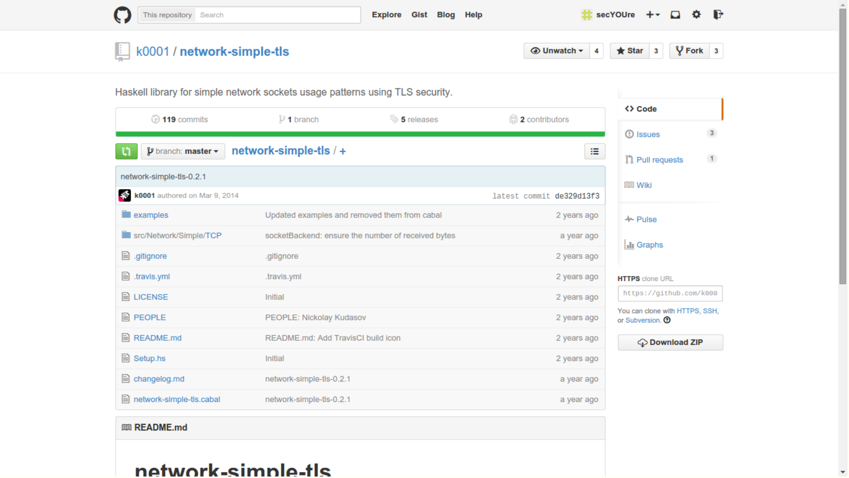
- “Insert vulnerabilities into commercial encryption systems, IT systems, networks and endpoint communications devices used by targets.”
- “influence policies, standard and specifications for commercial public key technologies.”
- Interfere with the supply-chain
- Disregard everything about policy
- Or, she is simply in the position to build the security module used by the Certification Authority for generating the key material

Three Modules

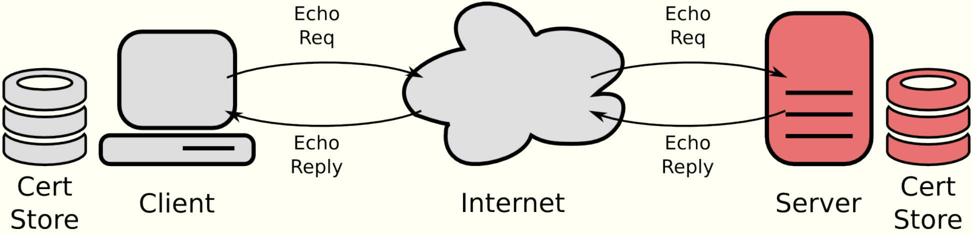




network-simple-tls



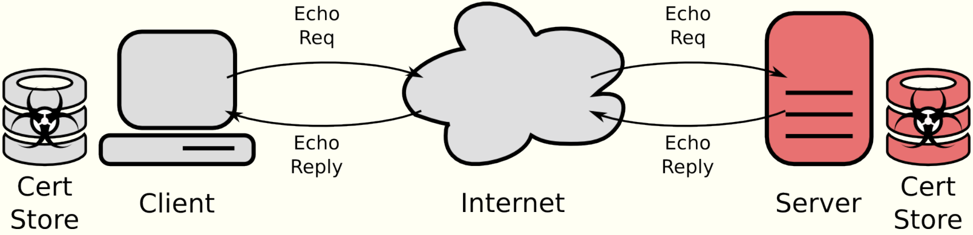
Echo service over TLS



Where is the backdoor?

If the client and server code is contributed by an open-source project and it is used *as-is*, where is the backdoor?

Where is the backdoor?



A Covert Channel

- The upper order bits of the RSA modulus encode the asymmetric encryption of a seed generated at random

A Covert Channel

- The upper order bits of the RSA modulus encode the asymmetric encryption of a seed generated at random
- The same seed was used to generate one of the RSA primes of the CA public-key modulus

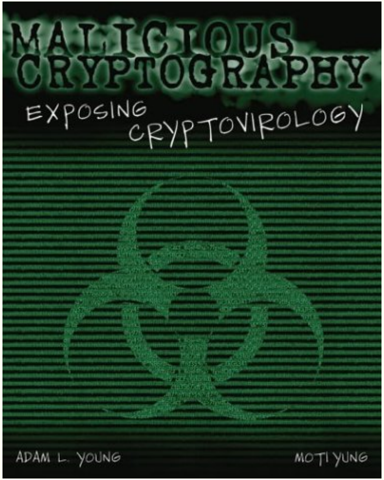
A Covert Channel

- The upper order bits of the RSA modulus encode the asymmetric encryption of a seed generated at random
- The same seed was used to generate one of the RSA primes of the CA public-key modulus
- The RSA modulus is at the same time a RSA public-key and an ciphertext that gives to the backdoor designer the ability to factor with ease the modulus

Where the backdoor is not

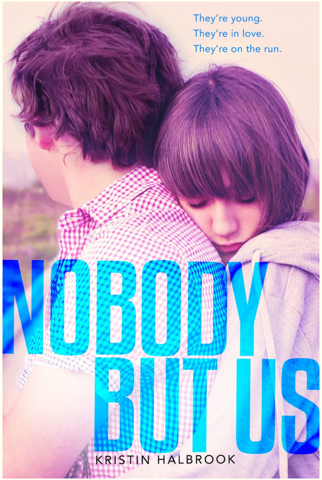
No backdoor was slipped into the cryptographic credentials issued to the communicating endpoints

SETUP Attacks



- Notion introduced by Adam Young and Moti Yung at Crypto ’96
- Young and Yung elliptic-curve asymmetric backdoor in RSA key generation
- Expands on ‘A Space Efficient Backdoor in RSA and its Applications’, Selected Areas in Cryptography ’05
- A working implementation at `http://cryptovirology.com`

NOBUS



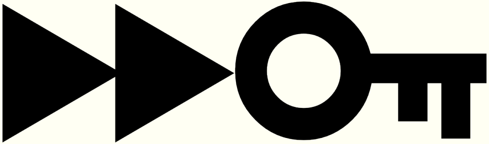
- The exploitation requires access to resources not embedded in the backdoor itself
- e.g., elliptic-curve private key
- The vulnerability can be exploited by the backdoor designer and by whoever gains access to the associated key-recovery system

How many of you believe that it is possible to forbid an enemy intelligence organization from gaining access to a private key?

Indistinguishability

- Assuming ECDDH holds
- The backdoor key pairs appear to all probabilistic polynomial time algorithms like genuine RSA key pairs
- Black-box access to the key-generator does not allow detection

Forward Secrecy



- If a reverse-engineer breaches the key-generator, then the previously stolen information remains confidential

Reusability

- The backdoor can be used multiple times and against multiple targets

Impact

A Subtle Attack

- Break TLS security guarantees at will

A Subtle Attack

- Break TLS security guarantees at will
- Impersonation (e.g., authentication failure)

A Subtle Attack

- Break TLS security guarantees at will
- Impersonation (e.g., authentication failure)
- Message tampering (e.g., integrity erosion)

A Subtle Attack

- Break TLS security guarantees at will
- Impersonation (e.g., authentication failure)
- Message tampering (e.g., integrity erosion)
- Active eavesdropping of encrypted communications (e.g., confidentiality loss)

A Subtle Attack

- Break TLS security guarantees at will
- Impersonation (e.g., authentication failure)
- Message tampering (e.g., integrity erosion)
- Active eavesdropping of encrypted communications (e.g., confidentiality loss)
- No need to have access to any private key used by system actors

A Subtle Attack

- Break TLS security guarantees at will
- Impersonation (e.g., authentication failure)
- Message tampering (e.g., integrity erosion)
- Active eavesdropping of encrypted communications (e.g., confidentiality loss)
- No need to have access to any private key used by system actors
- No need to tamper with the communicating endpoints

A Subtle Attack

- Break TLS security guarantees at will
- Impersonation (e.g., authentication failure)
- Message tampering (e.g., integrity erosion)
- Active eavesdropping of encrypted communications (e.g., confidentiality loss)
- No need to have access to any private key used by system actors
- No need to tamper with the communicating endpoints
- Need to retain control over the key-generation of the target RSA modulus

Is the malicious implementer a threat mitigated by IT product security certifications?

Fictional Security

A single CA certificate with a secretly embedded backdoor renders the entire TLS security fictional

One Rotten Apple...

One rotten apple...

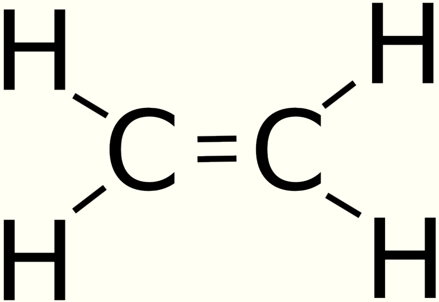
… spoils the whole barrel

… spoils the whole barrel

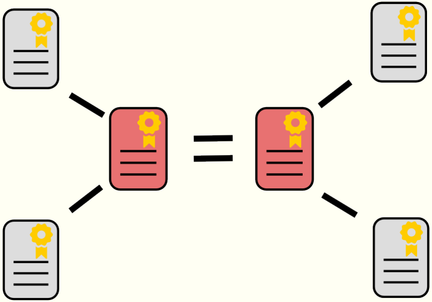
Ethylene

Universal implicit cross-certification is the ethylene of trust

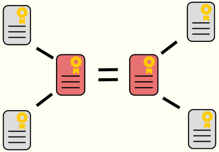
C2H4



Cross Certification

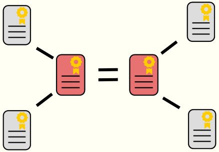


Cross Certification



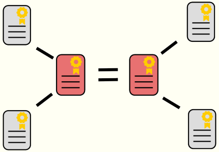
- Cross certification enables entities in one public key infrastructure to trust entities in another PKI

Cross Certification



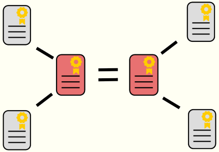
- Cross certification enables entities in one public key infrastructure to trust entities in another PKI
- This mutual trust relationship should be typically supported by a cross-certification agreement between the CAs in each PKI

Cross Certification



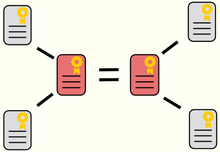
- Cross certification enables entities in one public key infrastructure to trust entities in another PKI
- This mutual trust relationship should be typically supported by a cross-certification agreement between the CAs in each PKI
- The agreement establishes the responsabilities and liability of each party

Explicit Cross Certification



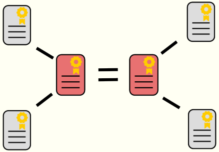
- Each CA is required to issue a certificate to the other to establish a relationship in both directions

Explicit Cross Certification

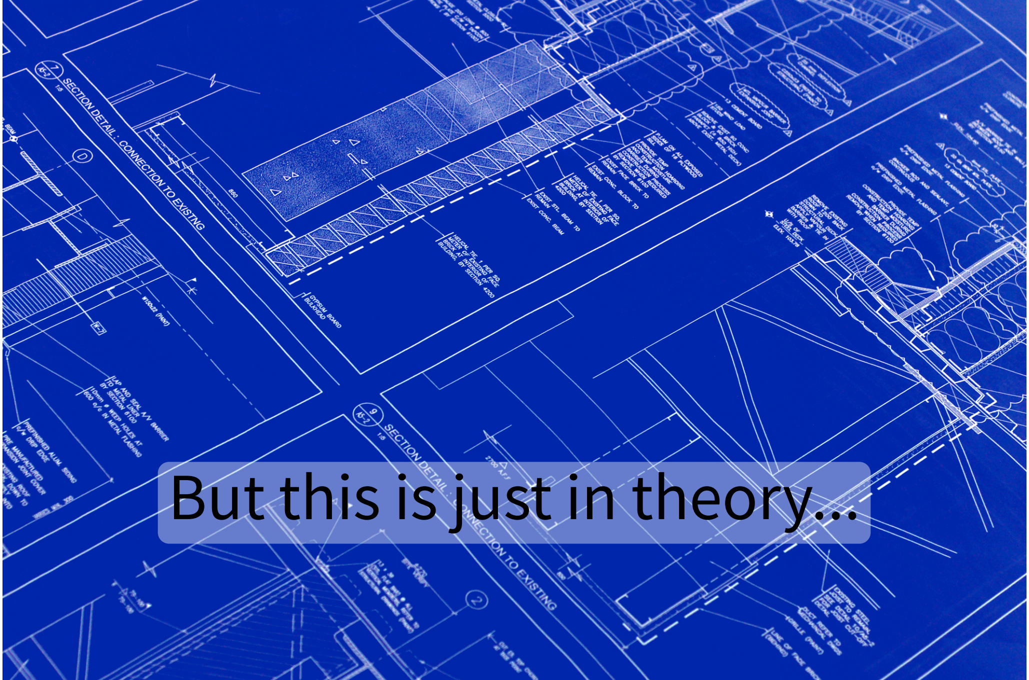


- Each CA is required to issue a certificate to the other to establish a relationship in both directions
- The path of trust is not hierarchical, although the separate PKIs may be certificate hierarchies

Explicit Cross Certification



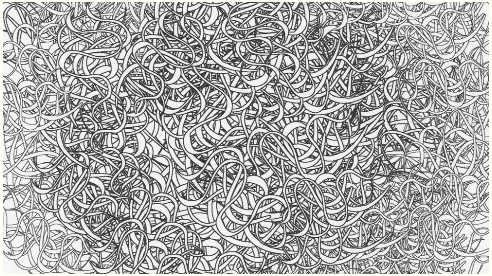
- Each CA is required to issue a certificate to the other to establish a relationship in both directions
- The path of trust is not hierarchical, although the separate PKIs may be certificate hierarchies
- After two CAs have established and specified the terms of trust and issued the certificates to each other, entities within the separate PKIs can interact subject to the policies specified in the certificates



But this is just in theory...

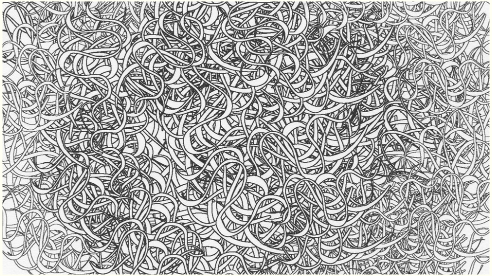
In practice:

Implicit Cross Certification



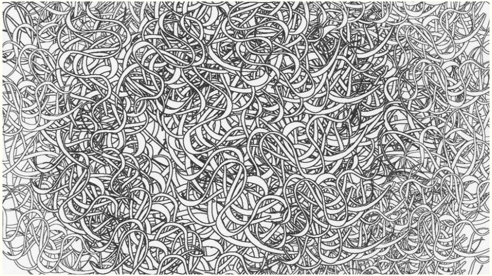
- Most current PKI software employs a form of implicit cross certification in which all root CAs are equally trusted

Implicit Cross Certification



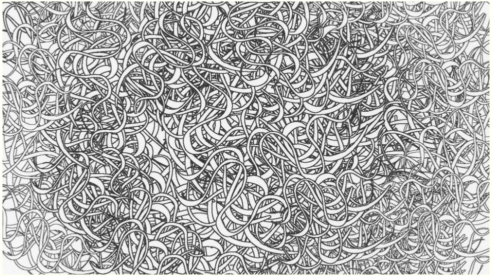
- Most current PKI software employs a form of implicit cross certification in which all root CAs are equally trusted
- Equivalent to unbounded cross certification among all CAs

Implicit Cross Certification



- Most current PKI software employs a form of implicit cross certification in which all root CAs are equally trusted
- Equivalent to unbounded cross certification among all CAs
- Any certificate can be trivially replaced by a masquereder's certificate from another CA

Implicit Cross Certification



- Most current PKI software employs a form of implicit cross certification in which all root CAs are equally trusted
- Equivalent to unbounded cross certification among all CAs
- Any certificate can be trivially replaced by a masquereder's certificate from another CA
- The security of any certificate is reduced to that of the least trustworthy CA, who can issue bogus certificate to usurp the legitimate one, at the same level of trust

CA Certificate in a MitM Proxy

Superfish Adware

PKI is Not Dead, Just Resting

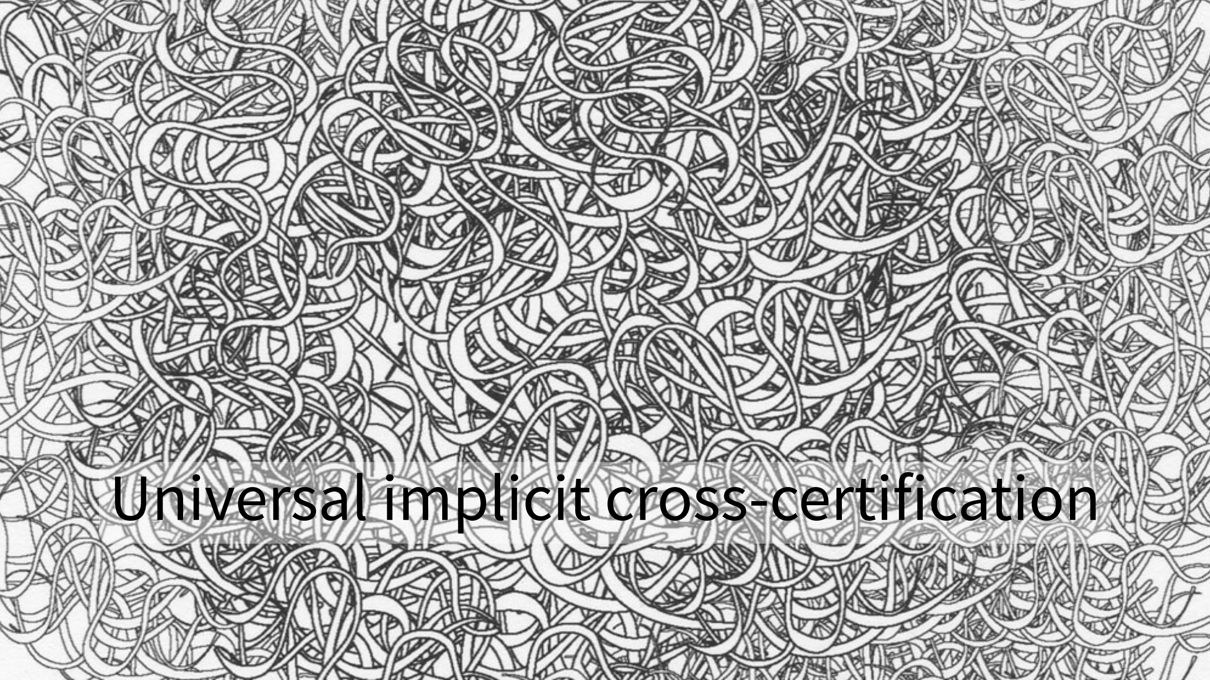
PKI: It's Not Dead, Just Resting

Peter Gutmann
University of Auckland

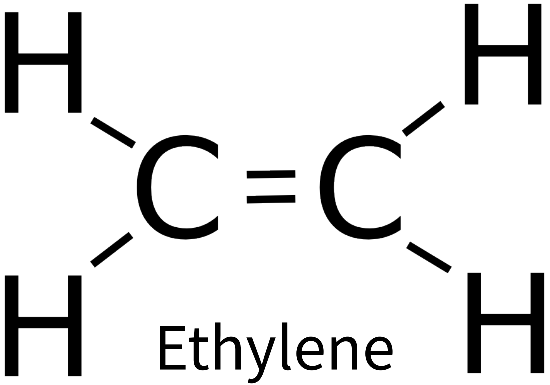
Abstract

Despite enthusiastic predictions in the trade press, an X.509-style PKI has so far failed to eventuate to any significant degree. This paper looks at some of the reasons behind this, examining why a pure X.509-style PKI may never appear outside a few closed, highly-controlled environments such as government agencies. On the other hand there are many instances in which situation- and application-specific uses of certificates can be employed in a manner that avoids the shortcomings of X.509's one-size-(mis)fits-all approach. The paper examines a number of these situation-specific approaches to working with certificates, and concludes with a collection of useful design rules to consider before embarking on a PKI project.

1. Introduction


Universal implicit cross-certification

Security

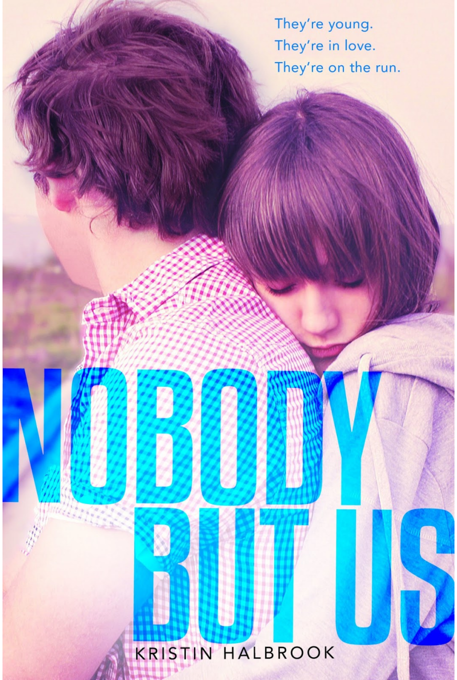


Ethylene


Rotting fruit


As weak as the weakest link

Multiple attackers attracted


They're young.
They're in love.
They're on the run.
NOBODY
BUT US
KRISTIN HALBROOK


HERGESTELLT VON DER DEUTSCHEN GRAMMOPHON GESELLSCHAFT · REGD · TR · M · ALLE HERSTELLER- UND URHEBERRECHTE VORBEHALTEN: ÜBERSPIELUNG, ÖFFENTLICHE AUFFÜHRUNG UND RUNDFUNKSENDUNG VERBOTEN.
Polydor
BIEM
N H
M
45
66 815 B
EVERYBODY ELSE BUT ME
(MacFarlane)
PETER KRAUS
Accompaniment by Bob Sharples


OMG
ME TOO
memegenerator.net

Negating any meaningful security whatsoever

It is essential to have assurance about the security of each implementation of vulnerable key-generation algorithm employed by trusted credential issuers

Hundreds CAs

211 CA certificates installed

Have we sufficient assurance about the hundreds CA certificates we daily entrust our business upon?

Requirements

- Publicly trusted certificates to be issued in compliance with European Standard EN 319 411-3

CAB CA/BROWSER FORUM

Requirements

- Publicly trusted certificates to be issued in compliance with European Standard EN 319 411-3
- CA key generation to be carried out within a device that meets the requirements identified by some approved PP

Requirements

- Publicly trusted certificates to be issued in compliance with European Standard EN 319 411-3
- CA key generation to be carried out within a device that meets the requirements identified by some approved PP
- CEN Workshop Agreement 14167, Part 2-3-4 are three of those PP

Requirements

CA/BROWSER FORUM

- Publicly trusted certificates to be issued in compliance with European Standard EN 319 411-3
- CA key generation to be carried out within a device that meets the requirements identified by some approved PP
- CEN Workshop Agreement 14167, Part 2-3-4 are three of those PP
- EAL4 Augmented

Requirements

- Publicly trusted certificates to be issued in compliance with European Standard EN 319 411-3
- CA key generation to be carried out within a device that meets the requirements identified by some approved PP
- CEN Workshop Agreement 14167, Part 2-3-4 are three of those PP
- EAL4 Augmented
- Augmentation from adherence to ADV_IMP.2, AVA_CCA.1, and AVA_VLA.4

ADV_IMP.2, AVA_CCA.1, and AVA_VLA.4

- Focused on assessing the vulnerabilities in the TOE

CA/BROWSER FORUM

ADV_IMP.2, AVA_CCA.1, and AVA_VLA.4

- Focused on assessing the vulnerabilities in the TOE
- Guaranteeing that the implementation representation is an accurate and complete instantiation of the TSF requirements

ADV_IMP.2, AVA_CCA.1, and AVA_VLA.4

CA/BROWSER FORUM

- Focused on assessing the vulnerabilities in the TOE
- Guaranteeing that the implementation representation is an accurate and complete instantiation of the TSF requirements
- Special emphasis on identifying covert channels and estimating their capacity

ADV_IMP.2, AVA_CCA.1, and AVA_VLA.4

- Focused on assessing the vulnerabilities in the TOE
- Guaranteeing that the implementation representation is an accurate and complete instantiation of the TSF requirements
- Special emphasis on identifying covert channels and estimating their capacity
- SETUP attacks makes use of the key-generation as a covert channel for itself

Yet

- Developer is in charge for the vulnerability assessment and documentation

Yet

- Developer is in charge for the vulnerability assessment and documentation
- Conflicts with our threat model

Yet

- Developer is in charge for the vulnerability assessment and documentation
- Conflicts with our threat model
- The evaluator is left with the documentation and the implementation representation to be assessed

Yet

- Developer is in charge for the vulnerability assessment and documentation
- Conflicts with our threat model
- The evaluator is left with the documentation and the implementation representation to be assessed
- Can the presence of backdoor can be ruled out at the required assurance level?

Yet

- Developer is in charge for the vulnerability assessment and documentation
- Conflicts with our threat model
- The evaluator is left with the documentation and the implementation representation to be assessed
- Can the presence of backdoor can be ruled out at the required assurance level?
- Formal methods required only at the two highest levels (EAL6 and EAL7)

Yet

- Developer is in charge for the vulnerability assessment and documentation
- Conflicts with our threat model
- The evaluator is left with the documentation and the implementation representation to be assessed
- Can the presence of backdoor can be ruled out at the required assurance level?
- Formal methods required only at the two highest levels (EAL6 and EAL7)
- Implementation representation may render backdoor detection unlikely (e.g., HDL at design time, netlist at fabrication time)

Key Takeaway

As long as the implementations of RSA — or, more generally, algorithms vulnerable to this class of attacks — used by trusted entities (e.g., CA) cannot be audited by relying parties (e.g., x.509 end-entities), any trust-anchor for the same trusted entities (e.g., root certificate) is to be regarded as a potential backdoor

Key Takeaway - Ctd

As long as the implementation of algorithms adopted by CAs and vulnerable to this class of backdoors cannot be audited by relying parties, the assurance provided by illusoryTLS (i.e., none whatsoever) is not any different from the assurance provided by systems relying upon TLS and RSA certificates for origin authentication, confidentiality, and message integrity guarantees

Mitigations

- Key Pinning, RFC 7469, Public Key Pinning Extension for HTTP (HPKP), April 2015
- Certificate Transparency, RFC 6962, June 2013
- DANE, DNS-based Authentication of Named Entities, RFC 6698, August 2012
- Tack, Trust Assertions for Certificate Keys, draft-perrin-tls-tack-02.txt, Expired
- Proper explicit cross-certification

A Backdoor Embedding Algorithm

Subtleness

The subtleness of a backdoor planted in a cryptographic credential resides in the *absence of malicious logic* in the system whose security it erodes.

An attack variant

RyanC — `https://gist.github.com/ryancdotorg/18235723e926be0afbdd`

Idea

1. Embed a Curve25519 public-key into the key-generator

Idea

1. Embed a Curve25519 public-key into the key-generator
2. Generate an ephemeral Curve25519 key at random

Idea

1. Embed a Curve25519 public-key into the key-generator
2. Generate an ephemeral Curve25519 key at random
3. Compute a shared secret using Elliptic Curve Diffie-Hellman

Idea

1. Embed a Curve25519 public-key into the key-generator
2. Generate an ephemeral Curve25519 key at random
3. Compute a shared secret using Elliptic Curve Diffie-Hellman
4. Use the shared secret to seed at cryptographically secure pseudo-random number generator (CSPRNG) based on AES run in CTR mode

Idea

1. Embed a Curve25519 public-key into the key-generator
2. Generate an ephemeral Curve25519 key at random
3. Compute a shared secret using Elliptic Curve Diffie-Hellman
4. Use the shared secret to seed at cryptographically secure pseudo-random number generator (CSPRNG) based on AES run in CTR mode
5. Generate a normal RSA key using the seeded CSPRNG

Idea

1. Embed a Curve25519 public-key into the key-generator
2. Generate an ephemeral Curve25519 key at random
3. Compute a shared secret using Elliptic Curve Diffie-Hellman
4. Use the shared secret to seed at cryptographically secure pseudo-random number generator (CSPRNG) based on AES run in CTR mode
5. Generate a normal RSA key using the seeded CSPRNG
6. Replace 32-bytes of the generated modulus with the ephemeral Curve25519 public-key

Idea

1. Embed a Curve25519 public-key into the key-generator
2. Generate an ephemeral Curve25519 key at random
3. Compute a shared secret using Elliptic Curve Diffie-Hellman
4. Use the shared secret to seed at cryptographically secure pseudo-random number generator (CSPRNG) based on AES run in CTR mode
5. Generate a normal RSA key using the seeded CSPRNG
6. Replace 32-bytes of the generated modulus with the ephemeral Curve25519 public-key
7. Use the original prime factors to compute two new primes leading to a new modulus embedding the ephemeral public-key

Idea

1. Embed a Curve25519 public-key into the key-generator
2. Generate an ephemeral Curve25519 key at random
3. Compute a shared secret using Elliptic Curve Diffie-Hellman
4. Use the shared secret to seed at cryptographically secure pseudo-random number generator (CSPRNG) based on AES run in CTR mode
5. Generate a normal RSA key using the seeded CSPRNG
6. Replace 32-bytes of the generated modulus with the ephemeral Curve25519 public-key
7. Use the original prime factors to compute two new primes leading to a new modulus embedding the ephemeral public-key
8. Output the RSA key with the secretly embedded backdoor

Key Recovery

1. Extracts the ephemeral Curve25519 public-key from the target modulus

Key Recovery

1. Extracts the ephemeral Curve25519 public-key from the target modulus
2. Computes the shared secret via ECDH and using the private-key associated to the public-key embedded in the key generator

Key Recovery

1. Extracts the ephemeral Curve25519 public-key from the target modulus
2. Computes the shared secret via ECDH and using the private-key associated to the public-key embedded in the key generator
3. Uses the shared secret to seed the CSPRNG based on AES run in CTR mode

Key Recovery

1. Extracts the ephemeral Curve25519 public-key from the target modulus
2. Computes the shared secret via ECDH and using the private-key associated to the public-key embedded in the key generator
3. Uses the shared secret to seed the CSPRNG based on AES run in CTR mode
4. Generates a normal RSA key using the seeded CSPRNG

Key Recovery

1. Extracts the ephemeral Curve25519 public-key from the target modulus
2. Computes the shared secret via ECDH and using the private-key associated to the public-key embedded in the key generator
3. Uses the shared secret to seed the CSPRNG based on AES run in CTR mode
4. Generates a normal RSA key using the seeded CSPRNG
5. Replaces 32-bytes of the generated modulus with the ephemeral Curve25519 public-key

Key Recovery

1. Extracts the ephemeral Curve25519 public-key from the target modulus
2. Computes the shared secret via ECDH and using the private-key associated to the public-key embedded in the key generator
3. Uses the shared secret to seed the CSPRNG based on AES run in CTR mode
4. Generates a normal RSA key using the seeded CSPRNG
5. Replaces 32-bytes of the generated modulus with the ephemeral Curve25519 public-key
6. Uses the original prime factors to compute two new primes leading to the target modulus embedding the ephmeral public-key

Key Recovery

1. Extracts the ephemeral Curve25519 public-key from the target modulus
2. Computes the shared secret via ECDH and using the private-key associated to the public-key embedded in the key generator
3. Uses the shared secret to seed the CSPRNG based on AES run in CTR mode
4. Generates a normal RSA key using the seeded CSPRNG
5. Replaces 32-bytes of the generated modulus with the ephemeral Curve25519 public-key
6. Uses the original prime factors to compute two new primes leading to the target modulus embedding the ephmeral public-key
7. Output the recovered RSA private key

Broken

- Although the idea is nice
- The key pairs generated using this algorithm fall short in terms of indistiguishability
- It is easy to tell backdoored certificates apart from genuine RSA certificate using only black-box access

Does anybody see why this is the case?

Distinguishing Attack

- A public-key embedded into an RSA modulus

Distinguishing Attack

- A public-key embedded into an RSA modulus
- Elliptic curve public-keys are points on the curve

Distinguishing Attack

- A public-key embedded into an RSA modulus
- Elliptic curve public-keys are points on the curve
- And elliptic curve points are easily distinguished from uniform random strings

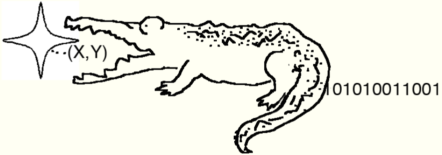
Distinguishing Attack

- A public-key embedded into an RSA modulus
- Elliptic curve public-keys are points on the curve
- And elliptic curve points are easily distinguished from uniform random strings
- A security evaluator could check if the coordinates encoded using the candidate 32-byte substrings of the modulus satisfy the elliptic curve equation

Repairing the Backdoor

If we could make the elliptic curve points indistinguishable from random strings, then the backdoor indistinguishability would be retained

Elligator



- Censorship sucks!
- Daniel J. Bernstein, Anna Krasnova, Mike Hamburg, Tanja Lange
- an encoding for points on a single curve as strings indistiguishable from uniform random strings
- `http://elligator.cr.yp.to`

Inherently Dual Use

All cyber security technology is inherently dual use

Undetectability for Good or Ill

- Just like any and all cyber security tools
- Undetectability of curve points can be used for good or ill
- For censorship-circumvention or surveillance

Between Offense and Defense

I believe we can positively contribute to the discussion and practice of information security by walking the fine line between offense and defense

Code

- Website — `http://illusorytls.com`
- illusoryTLS — `https://github.com/secYOUre/illusoryTLS`
- pyelligator — `https://github.com/secYOUre/pyelligator`
- rsaelligatorbd — `https://github.com/secYOUre/rsaelligatorbd`

Elligator backdoor embedding

- Embed a Curve25519 public-key into the key-generator

```
MASTER_PUB_HEX = '525e422e42c9c662362a7326c3c5c785ac7ef52e86782c4ac3c06887583e7a6f'
master_pub = unhexlify(MASTER_PUB_HEX)
```

Elligator backdoor embedding

- Generate an ephemeral Curve25519 key at random and the associated uniform representative string

```
while True:
   private = urandom(32)
   (v, pub, rep) = elligator.scalarbasemult(private)
   if v:
       break
```

Elligator backdoor embedding

- Compute a shared secret using ECDH
- Use the shared secret to seed a CSPRNG based on AES run in CTR mode

```
# combine the ECDH keys to generate the seed
seed = nacl.crypto_box_beforenm(master_pub, private)

prng = AESPRNG(seed)
```

Elligator backdoor embedding

- Generate a normal RSA key using the seeded CSPRNG

```
    # deterministic key generation from seed
    rsa = build_key(embed=rep, pos=80, randfunc=prng.randbytes)
    ...
def build_key(bits=2048, e=65537, embed='', pos=1, randfunc=None):
    # generate base key
    rsa = RSA.generate(bits, randfunc)
```

Elligator backdoor embedding

- Replace 32-bytes of the generated modulus with the representative string associated to the ephemeral Curve25519 public-key

```
  # extract modulus as a string
  n_str = unhexlify(str(hex(rsa.n))[2:-1])
  # embed data into the modulus
  n_hex = hexlify(replace_at(n_str, embed, pos))
  ...
# overwrite some bytes in orig at a specificed offset
def replace_at(orig, replace, offset):
  return orig[0:offset] + replace + orig[offset+len(replace):]
```

Elligator backdoor embedding

- Use the original prime factors to compute to new primes leading to a new modulus embedding the uniform representative string

```
n = gmpy.mpz(n_hex, 16)
p = rsa.p
# compute a starting point to look for a new q value
pre_q = n / p
# use the next prime as the new q value
q = pre_q.next_prime()
n = p * q
phi = (p-1) * (q-1)
# compute new private exponent
d = gmpy.invert(e, phi)
# make sure that p is smaller than q
if p > q:
   (p, q) = (q, p)
```

Elligator backdoor embedding

- Output the backdoored RSA key

```
return RSA.construct((long(n), long(e), long(d), long(p), long(q)))
```

Key Recovery

- Extracts the representative string from the target modulus

```
    #Load an x.509 certificate from a file
    x509 = X509.load_cert(sys.argv[2])
    # Pull the modulus out of the certificate
    orig_modulus = unhexlify(x509.get_pubkey().get_modulus())
    (seed, rep) = recover_seed(key=sys.argv[1], modulus=orig_modulus, pos=80)
    ...
def recover_seed(key='', modulus=None, pos=1):
    ...
    rep = modulus[pos:pos+32]
```

Key Recovery

- Maps the representative string to the candidate ephemeral Curve25519 public-key

```
pub = elligator.representativetopublic(rep)
```

Key Recovery

- Computes the shared secret via ECDH and using the private-key associated to the public-key embedded in the key-generator
- Uses the shared secret to seed the CSPRNG based on AES run in CTR mode

```
def recover_seed(key='', modulus=None, pos=1):
   # recreate the master private key from the passphrase
   master = sha256(key).digest()
   ...
   # compute seed with master private and ephemeral public key
   return (nacl.crypto_box_beforenm(pub, master), rep)

   ...
   (seed, rep) = recover_seed(key=sys.argv[1], modulus=orig_modulus, pos=80)
   prng = AESPRNG(seed)
```

Key Recovery

- Generates a normal RSA key using the seeded CSPRNG

```
    # deterministic key generation from seed
    rsa = build_key(embed=rep, pos=80, randfunc=prng.randbytes)
    ...
def build_key(bits=2048, e=65537, embed='', pos=1, randfunc=None):
    # generate base key
    rsa = RSA.generate(bits, randfunc)
```

Key Recovery

- Replaces 32-bytes of the generated modulus with the representative string found in the target modulus

```
# extract modulus as a string
n_str = unhexlify(str(hex(rsa.n))[2:-1])
# embed data into the modulus
n_hex = hexlify(replace_at(n_str, embed, pos))
```

Key Recovery

- Uses the original prime factors to compute two new primes leading to the target modulus embedding the uniform representative string

```
n = gmpy.mpz(n_hex, 16)
p = rsa.p
# compute a starting point to look for a new q value
pre_q = n / p
# use the next prime as the new q value
q = pre_q.next_prime()
n = p * q
phi = (p-1) * (q-1)
# compute new private exponent
d = gmpy.invert(e, phi)
# make sure that p is smaller than q
if p > q:
  (p, q) = (q, p)
```

Key Recovery

- Output the recovered RSA key

```
return RSA.construct((long(n), long(e), long(d), long(p), long(q)))
...
print rsa.exportKey()
```

Demo

Conclusions


Vincent

Though I am often in the depths of misery, there is still calmness, pure harmony and music inside me.

Vincent van Gogh

> *Though we are often in the depths of insecurity, there is still calmness, pure harmony and music inside us.*

THANK YOU

Questions?

Backup

Normal RSA Key Generation — Young and Yung

1. Let e be the public RSA exponent (e.g., $2^{16}+1$)
2. Choose a large number p randomly (e.g., 1024 bits long)
3. If p is composite or $gcd(e, p-1) \neq 1$ then goto to step 1
4. Choose a large number q randomly (e.g., 1024 bits long)
5. If q is composite or $gcd(e, p-1) \neq 1$ then goto to step 3
6. Output the public-key $(N = pq, e)$ and the private-key p
7. The private exponent d is found by solving for (d, k) in $ed + k\phi(n) = 1$ using the extended Euclidean algorithm

RSA Encryption/Decryption — Young and Yung

- $N = p * q$, where p and q are large primes known to the key owner
- Everyone knows N and e
- Let d be a privete key exponent where $ed = 1 mod (p-1)(q-1)$
- To encrypt $m \in Z_n^*$ (after padding) compute: $c = m^e modN$
- To decrypt the ciphertext c compute: $m = c^d modN$
- As far as we know: Only with known factorization given N and e, one can find d

Elliptic Curve Decision Diffie-Hellman Problem

- Let C an elliptic-curve equation over the finite field $\mathbb{F}_q$ with prime order n
- Let G be the base point of the curve
- Given three point elements (xG), (yG) and (zG)
- Decide whether $(zG = xyG)$, or not
- Where (x, y, z) are chosen randomly and $1 < x, y, z < n$