Introduction to cryptographic protocols

Bruno Blanchet

Inria Paris
Bruno.Blanchet@inria.fr

Year 2016-17

(Partly based on slides by Stéphanie Delaune)
Overview

1. Protocols

2. Primitives

3. Verification
   - Symbolic model
   - Computational model

4. Ex: credit card

5. Ex: Needham-Schroeder

6. Ex: TLS
Cryptographic protocols

- small programs designed to **secure** communication (various security goals)
- use **cryptographic primitives** (e.g. encryption, hash function,
Cryptographic protocols

- small programs designed to secure communication (various security goals)
- use cryptographic primitives (e.g. encryption, hash function,
Security properties (1)

- **Secrecy**: May an intruder learn some secret message between two honest participants?

- **Authentication**: Is the agent Alice really talking to Bob?

- **Fairness**: Alice and Bob want to sign a contract. Alice initiates the protocol. May Bob obtain some advantage?

- **Non-repudiation**: Alice sends a message to Bob. Alice cannot later deny having sent this message. Bob cannot deny having received the message.

...
Security properties: E-voting (2)

Eligibility: only legitimate voters can vote, and only once

Fairness: no early results can be obtained which could influence the remaining voters

Individual verifiability: a voter can verify that her vote was really counted

Universal verifiability: the published outcome really is the sum of all the votes
Privacy: the fact that a particular voter voted in a particular way is not revealed to anyone

Receipt-freeness: a voter cannot prove that she voted in a certain way (this is important to protect voters from coercion)

Coercion-resistance: same as receipt-freeness, but the coercer interacts with the voter during the protocol, (e.g. by preparing messages)
Cryptographic primitives

Algorithms that are frequently used to build computer security systems. These routines include, but are not limited to, encryption and signature functions.
Cryptographic primitives

Algorithms that are frequently used to build computer security systems. These routines include, but are not limited to, encryption and signature functions.

Symmetric encryption

Examples: Caesar encryption, DES, AES, ...
Cryptographic primitives

Algorithms that are frequently used to build computer security systems. These routines include, but are not limited to, encryption and signature functions.

Asymmetric encryption

- Encryption using public key
- Decryption using private key
Cryptographic primitives

Algorithms that are frequently used to build computer security systems. These routines include, but are not limited to, encryption and signature functions.

Signature

- Signature
- Verification
- Private key
- Public key
Why verify security protocols?

The verification of security protocols has been and is still a very active research area.

- Their design is error prone.
- Security errors are not detected by testing: they appear only in the presence of an adversary.
- Errors can have serious consequences.
Active attacker:

- the attacker can intercept all messages sent on the network
- he can compute messages
- he can send messages on the network
The **symbolic model** or “Dolev-Yao model” is due to Needham and Schroeder [1978] and Dolev and Yao [1983].

- The cryptographic primitives are **blackboxes**.
- The messages are **terms** on these primitives.
  - $\{m\}_k$ encryption of the message $m$ with key $k$,
  - $(m_1, m_2)$ pairing of messages $m_1$ and $m_2$, …
- The attacker is restricted to compute only using these primitives.
  - $\Rightarrow$ perfect cryptography assumption

One can add equations between primitives, but in any case, one makes the hypothesis that the only equalities are those given by these equations.

This model makes automatic proofs relatively easy (AVISPA, ProVerif, …).
The computational model has been developed at the beginning of the 1980’s by Goldwasser, Micali, Rivest, Yao, and others.

- The messages are bitstrings.
- The cryptographic primitives are functions on bitstrings.
- The attacker is any probabilistic (polynomial-time) Turing machine.

This model is much more realistic than the symbolic model, but until recently proofs were only manual.
The **computational model** is still just a **model**, which does not exactly match reality.

In particular, it ignores **side channels**:

- timing
- power consumption
- noise
- physical attacks against smart cards

which can give additional information.

In this course, we will mostly ignore side channels.
Symbolic model: example of attacks, replay attacks

transfer 100 euros into the merchant's account
Symbolic model: example of attacks, replay attacks

transfer 100 euros into
the merchant’s account

Bruno Blanchet (Inria)
Introduction to cryptographic protocols
Year 2016-17
Symbolic model: example of attacks, replay attacks

transfer 100 euros into the merchant's account

transfer 100 euros into the merchant's account

transfer 100 euros into the merchant's account

transfer 100 euros into the merchant's account

Bruno Blanchet (Inria)
Symbolic model: example of attacks, replay attacks

Example: attack on the decoders (TV)
→ block the message that cancels the subscription
Compute the set of all terms that the attacker can obtain.

This set is infinite:
- The attacker can generate messages of unbounded size.
- The number of sessions of the protocol is unbounded.
Complexity

- Bounded messages and number of sessions
  - $\Rightarrow$ finite state
  - Model checking: FDR [Lowe, TACAS’96]
- Bounded number of sessions but unbounded messages
  - $\Rightarrow$ insecurity is typically NP-complete
  - Constraint solving: Cl-AtSe, integrated in AVISPA
    Extensions of model checking: OFMC, integrated in AVISPA
- Unbounded messages and number of sessions
  - $\Rightarrow$ the problem is undecidable
    (proved later in this course)
Solutions to undecidability

- Rely on user interaction
  - Interactive theorem proving, Isabelle [Paulson, JCS’98]

- Use approximations
  - Abstract interpretation [Monniaux, SCP’03], TA4SP integrated in AVISPA
  - Typing [Abadi, JACM’99], [Gordon, Jeffrey, CSFW’02]
    (Sometimes also relies on type annotations by the user.)

- Allow non-termination

ProVerif uses approximations and allows non-termination.
Relevance of the symbolic model

- Numerous attacks have already been obtained.
- An attack in the symbolic model immediately implies one in the computational model (and a practical attack).
  - A proof in the symbolic model does not always imply a proof in the computational model (see next).
- Allows us to perform automatic verification.
Proofs in the computational model

- **Manual proofs by cryptographers:**
  - proofs by sequences of games [Shoup, Bellare & Rogaway]
    (See David Pointcheval’s course)

- **Automation:**
  - CryptoVerif
  - CertiCrypt, framework within Coq
  - EasyCrypt
  - Typing
Link between the two models

Computational soundness theorems:

Proof in the symbolic model $\Rightarrow$ proof in the computational model

modulo additional assumptions.

Approach pioneered by Abadi & Rogaway [2000]; many works since then.
Link between the two models: application

- **Indirect approach** to automating computational proofs:
  1. Automatic symbolic protocol verifier
  2. Computational proof in the symbolic model → computational model
Credit Card Payment Protocol
Example: credit card payment

- The **client** $C_l$ puts his credit card $C$ in the terminal $T$.
- The **merchant** enters the amount $M$ of the sale.
- The **terminal** authenticates the credit card.
- The **client** enters his PIN.
  - If $M \geq 100\,\text{€}$, then in 20% of cases,
    - The **terminal** contacts the bank $B$.
    - The **bank** gives its authorisation.
the Bank $B$, the Client $Cl$, the Credit Card $C$ and the Terminal $T$
the Bank $B$, the Client $C_l$, the Credit Card $C$ and the Terminal $T$

**Bank**

- a private signature key – $\text{priv}(B)$
- a public key to verify a signature – $\text{pub}(B)$
- a secret key shared with the credit card – $K_{CB}$
the Bank $B$, the Client $C_l$, the Credit Card $C$ and the Terminal $T$

**Bank**
- a private signature key – $\text{priv}(B)$
- a public key to verify a signature – $\text{pub}(B)$
- a secret key shared with the credit card – $K_{CB}$

**Credit Card**
- some $Data$: name of the cardholder, expiry date …
- a signature of the $Data$ – $\{\text{hash}(Data)\}_{\text{priv}(B)}$
- a secret key shared with the bank – $K_{CB}$
the Bank $B$, the Client $C_l$, the Credit Card $C$ and the Terminal $T$

**Bank**
- a private signature key – $\text{priv}(B)$
- a public key to verify a signature – $\text{pub}(B)$
- a secret key shared with the credit card – $K_{CB}$

**Credit Card**
- some $Data$: name of the cardholder, expiry date …
- a signature of the $Data$ – $\{\text{hash}(Data)\}_{\text{priv}(B)}$
- a secret key shared with the bank – $K_{CB}$

**Terminal**
- the public key of the bank – $\text{pub}(B)$
the terminal $T$ reads the credit card $C$:

1. $C \rightarrow T : \text{Data}, \{\text{hash(Data)}\}_{\text{priv}(B)}$
the terminal $T$ reads the credit card $C$:

1. $C \rightarrow T : \text{Data, } \{\text{hash(Data)}\}_{\text{priv}(B)}$

the terminal $T$ asks the code:

2. $T \rightarrow Cl : \text{code?}$
3. $Cl \rightarrow C : 1234$
4. $C \rightarrow T : \text{ok}$
Payment protocol

the terminal $T$ reads the credit card $C$:

1. $C \rightarrow T : \text{Data, } \{\text{hash(Data)}\}_{\text{priv}(B)}$

the terminal $T$ asks the code:

2. $T \rightarrow Cl : \text{code?}$
3. $Cl \rightarrow C : 1234$
4. $C \rightarrow T : \text{ok}$

the terminal $T$ requests authorisation from the bank $B$:

5. $T \rightarrow B : \text{auth?}$
6. $B \rightarrow T : 4528965874123$
7. $T \rightarrow C : 4528965874123$
8. $C \rightarrow T : \{4528965874123\}_{K_{CB}}$
9. $T \rightarrow B : \{4528965874123\}_{K_{CB}}$
10. $B \rightarrow T : \text{ok}$
Attack against credit cards

Initially, security was guaranteed by:

- cards hard to replicate,
- secrecy of keys and protocol.
Initially, security was guaranteed by:

- cards hard to replicate,
- secrecy of keys and protocol.

However, there are attacks!

- **cryptographic** attack: 320-bit keys are no longer secure,
- **logical** attack: no link between the 4-digit PIN code and the authentication,
- **hardware** attack: replication of cards.

The « YesCard »: how does it work?

Logical attack

1. $C \rightarrow T : \text{Data, } \{\text{hash(Data)}\}_{\text{priv}(B)}$
2. $T \rightarrow C I : \text{PIN?}$
3. $C I \rightarrow C : 1234$
4. $C \rightarrow T : \text{ok}$
The « YesCard »: how does it work?

Logical attack

1. $C \rightarrow T$ : $\text{Data, } \{\text{hash(Data)}\}_{\text{priv(B)}}$

2. $T \rightarrow Cl$ : $\text{PIN?}$

3. $Cl \rightarrow C'$ : 2345

4. $C' \rightarrow T$ : $\text{ok}$
The « YesCard »: how does it work?

Logical attack

1. $C \rightarrow T : Data, \{\text{hash}(Data)\}_{\text{priv}(B)}$
2. $T \rightarrow Cl : PIN?$
3. $Cl \rightarrow C' : 2345$
4. $C' \rightarrow T : ok$

Remark: there is always somebody to debit.
→ add a fake ciphertext on a fake card (Serge Humpich).
The « YesCard »: how does it work?

Logical attack

1. \( C \rightarrow T \) : \( \text{Data}, \{\text{hash(Data)}\}_{\text{priv}(B)} \)
2. \( T \rightarrow Cl \) : \( \text{PIN?} \)
3. \( Cl \rightarrow C' \) : 2345
4. \( C' \rightarrow T \) : ok

Remark: there is always somebody to debit.
→ add a fake ciphertext on a fake card (Serge Humpich).

1. \( C' \rightarrow T \) : \( XXX, \{\text{hash(XXX)}\}_{\text{priv}(B)} \)
2. \( T \rightarrow Cl \) : \( \text{PIN?} \)
3. \( Cl \rightarrow C' \) : 0000
4. \( C' \rightarrow T \) : ok
Needham-Schroeder (public-key) Protocol
Needham-Schroeder’s Protocol (1978)

- $A \rightarrow B : \{A, N_a\}_{\text{pub}(B)}$
- $B \rightarrow A : \{N_a, N_b\}_{\text{pub}(A)}$
- $A \rightarrow B : \{N_b\}_{\text{pub}(B)}$

Questions

- Is $N_b$ secret between $A$ and $B$?
- When $B$ receives $\{N_b\}_{\text{pub}(B)}$, does this message really come from $A$?

Attack.

An attack was found 17 years after its publication! \cite{Lowe96}
Needham-Schroeder’s Protocol (1978)

\[
\begin{align*}
A &\rightarrow B : \quad \{A, N_a\}_{\text{pub}(B)} \\
B &\rightarrow A : \quad \{N_a, N_b\}_{\text{pub}(A)} \\
A &\rightarrow B : \quad \{N_b\}_{\text{pub}(B)}
\end{align*}
\]

Questions

Is \(N_b\) secret between \(A\) and \(B\)?

When \(B\) receives \(\{N_b\}_{\text{pub}(B)}\), does this message really come from \(A\)?

Attack

An attack was found 17 years after its publication! [Lowe 96]
Needham-Schroeder’s Protocol (1978)

\[
\begin{align*}
A & \rightarrow B : \{A, N_a\}_{\text{pub}(B)} \\
B & \rightarrow A : \{N_a, N_b\}_{\text{pub}(A)} \\
\bullet A & \rightarrow B : \{N_b\}_{\text{pub}(B)}
\end{align*}
\]

Questions

Is \(N_b\) secret between \(A\) and \(B\)?

When \(B\) receives \(\{N_b\}_{\text{pub}(B)}\), does this message really come from \(A\)?

Attack.

An attack was found 17 years after its publication! [Lowe 96]
Needham-Schroeder’s Protocol (1978)

\[ A \rightarrow B : \{ A, N_a \}_{\text{pub}(B)} \]
\[ B \rightarrow A : \{ N_a, N_b \}_{\text{pub}(A)} \]
\[ A \rightarrow B : \{ N_b \}_{\text{pub}(B)} \]

Questions

Is \( N_b \) secret between \( A \) and \( B \)?

When \( B \) receives \( \{ N_b \}_{\text{pub}(B)} \), does this message really comes from \( A \)?

Attack.

An attack was found 17 years after its publication! [Lowe 96]
Needham-Schroeder’s Protocol (1978)

\[ A \rightarrow B : \{ A, N_a \}_{\text{pub}(B)} \]
\[ B \rightarrow A : \{ N_a, N_b \}_{\text{pub}(A)} \]
\[ A \rightarrow B : \{ N_b \}_{\text{pub}(B)} \]

Questions

- Is \( N_b \) secret between \( A \) and \( B \)?
- When \( B \) receives \( \{ N_b \}_{\text{pub}(B)} \), does this message really come from \( A \)?
Needham-Schroeder’s Protocol (1978)

\[ A \rightarrow B : \{ A, N_a \}_{pub(B)} \]
\[ B \rightarrow A : \{ N_a, N_b \}_{pub(A)} \]
\[ A \rightarrow B : \{ N_b \}_{pub(B)} \]

Questions

- Is \( N_b \) secret between \( A \) and \( B \) ?
- When \( B \) receives \( \{ N_b \}_{pub(B)} \), does this message really come from \( A \) ?

Attack

An attack was found 17 years after its publication! [Lowe 96]
Example: Man in the middle attack

Agent A  Intruder I  Agent B

Attack
- involving 2 sessions in parallel,
- an honest agent has to initiate a session with I.

\[ \begin{align*}
A & \rightarrow B : \{A, N_a\}_{\text{pub}(B)} \\
B & \rightarrow A : \{N_a, N_b\}_{\text{pub}(A)} \\
A & \rightarrow B : \{N_b\}_{\text{pub}(B)}
\end{align*} \]
Example: Man in the middle attack

\[ \{A, N_a\}_{\text{pub}(I)} \rightarrow \{A, N_a\}_{\text{pub}(B)} \]

Agent A \quad \text{Intruder I} \quad \text{Agent B}

\[
\begin{align*}
A & \rightarrow B : \{A, N_a\}_{\text{pub}(B)} \\
B & \rightarrow A : \{N_a, N_b\}_{\text{pub}(A)} \\
A & \rightarrow B : \{N_b\}_{\text{pub}(B)}
\end{align*}
\]
Example: Man in the middle attack

Agent A → B : $\{A, N_a\}_{\text{pub}(B)}$

B → A : $\{N_a, N_b\}_{\text{pub}(A)}$

A → B : $\{N_b\}_{\text{pub}(B)}$
Example: Man in the middle attack

Agent A → B : \(\{A, N_a\}_{\text{pub}(B)}\)
B → A : \(\{N_a, N_b\}_{\text{pub}(A)}\)
A → B : \(\{N_b\}_{\text{pub}(B)}\)
Example: Man in the middle attack

Agent A

\[ \{A, N_a\}_{\text{pub}(I)} \]
\[ \{N_a, N_b\}_{\text{pub}(A)} \]
\[ \{N_b\}_{\text{pub}(I)} \]

Intruder I

\[ \{N_a, N_b\}_{\text{pub}(A)} \]
\[ \{N_b\}_{\text{pub}(B)} \]

Agent B

\[ \{A, N_a\}_{\text{pub}(B)} \]
\[ \{N_a, N_b\}_{\text{pub}(A)} \]
\[ \{N_b\}_{\text{pub}(B)} \]

Attack

- the intruder knows \( N_b \),
- When B finishes his session (apparently with A), A has never talked with B.

\[ A \rightarrow B : \{A, N_a\}_{\text{pub}(B)} \]
\[ B \rightarrow A : \{N_a, N_b\}_{\text{pub}(A)} \]
\[ A \rightarrow B : \{N_b\}_{\text{pub}(B)} \]
Exercise

Propose a fix for the Needham-Schroeder protocol.

\begin{align*}
A \rightarrow B & : \{A, N_a\}_{\text{pub}(B)} \\
B \rightarrow A & : \{N_a, N_b\}_{\text{pub}(A)} \\
A \rightarrow B & : \{N_b\}_{\text{pub}(B)}
\end{align*}
Exercise: Denning-Sacco protocol

\[ A \rightarrow B : \{ \{ k \} \}_{sk_A} \{ \{ k \} \}_{pk_B} \quad k \text{ fresh} \]

Exercise

1. Find two attacks against this protocol. Which security properties do they break?
2. Propose fixes.

Bruno Blanchet (Inria)
Introduction to cryptographic protocols
Year 2016-17
Faille Heartbleed : les sites pour lesquels il est conseillé de changer son mot de passe

Le Monde.fr | 11.04.2014 à 04h48 • Mis à jour le 14.04.2014 à 10h08 | Par Michaël Szadkowski

Deux jours après la révélation d'une faille de sécurité au sein du protocole OpenSSL, baptisée « Heartbleed », cette dernière est décrite par certains comme « le pire cauchemar » qui puisse arriver concernant la sécurité des échanges sur Internet.

Le logiciel libre OpenSSL est installé sur les serveurs de très nombreux sites pour établir des connexions chiffrées et sécurisées entre ce dernier et ses utilisateurs. De très nombreux sites Internet utilisent OpenSSL pour sécuriser leurs échanges.
TLS: An implementation attack

Faille informatique « Heartbleed » : les leçons à tirer

La faille informatique baptisée Heartbleed est maintenant réparée. Mais les entreprises ont beaucoup à apprendre de ce malheureux épisode. La confidentialité des données de leurs clients a été mise en danger.

L’hémorragie est arrêtée. Entre mars 2012 et le 7 avril 2014, la faille informatique Heartbleed (« cœur qui saigne », en anglais) a concerné un très grand nombre de sites Internet, des réseaux sociaux en passant par les banques en ligne et les plate-formes de e-commerce. Le comble est que le danger provient justement du système chargé de sécuriser l’accès aux services sensibles comme le paiement en ligne...

L’alerte n’intervient que dans les premiers jours d’avril 2014. Jusqu’ici personne n’avaient constaté de péril. Le risque : des cyber-criminels pouvaient facilement retrouver les informations personnelles d’internautes utilisateurs de ces sites, dans la mémoire des serveurs informatiques. Horreur et mots de passe en premier lieu. Quelques jours après la
L'IMPACT DE LA FAILLE HEARTBLEED S'ÉTEND BIEN AU-DELÀ D'INTERNET

par Jim Finide

BOSTON (Reuters) - La faille de sécurité "Heartbleed" pourrait permettre à des pirates informatiques d'accéder à des boîtes mail, de contourner des pare-feu, voire de pirater des téléphones portables, selon des spécialistes en informatique qui ont prévenu jeudi que les risques pourraient s'étendre au-delà des seuls serveurs internet.

Cette vulnérabilité, qui semble déjà avoir été utilisée par ceux qui la maîtrisent, a été détectée dans une fonction qui permet à deux hosteurs de partager des informations.
The Heartbleed Hit List: The Passwords You Need to Change Right Now
TLS: An implementation attack

Anatomy of OpenSSL’s Heartbleed: Just four bytes trigger horror bug

The code behind the C-bomb dropped on the world

Analysis

The password-leaking OpenSSL bug dubbed Heartbleed is so bad, switching off the internet for a while sounds like a good plan.

A tiny flaw in the widely used encryption library allows anyone to trivially and secretly dip into vulnerable systems, from your bank’s HTTPS server to your private VPN, to steal passwords, login cookies, private crypto-keys and much more.

How, in 2014, is this possible?

A simple script for the exploit engine Metasploit can, in a matter of seconds, extract sensitive in-memory data from systems that rely on OpenSSL 1.0.1 to 1.0.1f for TLS encryption. The bug affects about 600,000, or 17.5 per cent, of trusted HTTPS websites, we’re told, as well as client software, email servers, chat services, and anything else using the aforementioned versions of OpenSSL.

A good number of popular web services have now been patched following disclosure of the vulnerability on Monday, you can use this tool to check (use at your own risk, of course), but don’t forget to do more than patch your OpenSSL installation if you’re affected – change your keys, dump your session cookies and evaluate your at-risk data.

Too long, didn’t read: A summary
Heartbeat: an extension of TLS that just checks that a server is alive.

- The client sends a packet to the server.
- The server replies to the client with the same packet.
Heartbleed explained (2): heartbeat message in detail

SSLv3 record

<table>
<thead>
<tr>
<th>Length of HeartbeatMessage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

HeartbeatMessage

<table>
<thead>
<tr>
<th>Type</th>
<th>Length of payload</th>
<th>Payload data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS1_HB_REQUEST</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Heartbleed explained (3): the attack

OpenSSL 1.0.1 before 1.0.1g does not check that the lengths are coherent!

**Heartbeat attack message**

SSLv3 record

<table>
<thead>
<tr>
<th>Length of HeartbeatMessage</th>
<th>4 bytes</th>
</tr>
</thead>
</table>

HeartbeatMessage

<table>
<thead>
<tr>
<th>Type</th>
<th>Length of payload</th>
<th>Payload data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS1_HB_REQUEST</td>
<td>65535 bytes</td>
<td>1 byte</td>
</tr>
</tbody>
</table>

**Heartbeat victim’s response**

SSLv3 record

<table>
<thead>
<tr>
<th>Length of HeartbeatMessage</th>
<th>65538 bytes</th>
</tr>
</thead>
</table>

HeartbeatMessage

<table>
<thead>
<tr>
<th>Type</th>
<th>Length of payload</th>
<th>Payload data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS1_HB_RESPONSE</td>
<td>65535 bytes</td>
<td>1 byte</td>
</tr>
</tbody>
</table>

65534 bytes leaked

Source: [http://www.theregister.co.uk/2014/04/09/heartbleed_explained/](http://www.theregister.co.uk/2014/04/09/heartbleed_explained/)
Verified implementations with F*, https://www.fstar-lang.org/

- F* is a new programming language
- ...putting together:
  - impure functional programming in ML
    - extracts to OCaml and F#, interoperates
  - the automation of SMT-based verification systems
    - like in Why3, Frama-C, Boogie, VCC, Dafny
  - the expressive power of interactive proof assistants based on dependent types
    - like in Coq, Agda, or Lean

(See Karthik Bhargavan and Catalin Hritcu’s course.)
miTLS, http://www.mitls.org/

- Formally verified reference implementation of TLS 1.2 in F7/F* (working towards TLS 1.3)
- Written from scratch focusing on verification
Lead to the discovery of many attacks in TLS implementations

SMACK: State Machine Attack

Implementations of the Transport Layer Security (TLS) protocol handle a variety of protocol versions, modes and key exchange methods, which can prescribe a different message sequence in the protocol stack. We address the problem of designing a robust security state machine that can correctly multiplex different protocol versions, modes.

Tracking the FREAK Attack

Good News! Your browser appears to be safe from the FREAK attack.

On Tuesday, March 3, 2015, researchers announced a new SSL/TLS vulnerability called the FREAK attack. It allows an attacker to intercept HTTPS connections between vulnerable clients and servers and force them to use weakened encryption to trick the implementations.

The FREAK attack was described in a research paper presented at the RSA Conference in San Francisco. The team of researchers includes engineers from NIST, Microsoft, and the University of Michigan. The team says the FREAK attack affects more than 10% of the world's internet traffic.

The BEAST Wins Again

Documents
- PDF of slides
- Summary of briefing for non-experts
- Paper: Virtual Heartbleed
- Paper: Triple Handshake

Exploit videos
- Disclaimer: The goal of these videos is not to harm or compromise anyone. The attacks are not real and are only used for educational purposes.

FREAK Attack Threatens SSL Clients

Posted by Soulskill on Tuesday March 03, 2015 @04:29PM
from the another-day-another-vuln dept.

For the nth time in the last couple of years, security experts are warning about Internet-scale vulnerability, this time in some popular SSL clients. The flaw allow attackers to force clients to downgrade to weakened cipher suites and break their supposedly encrypted communications through a man-in-the-middle attack.

Researchers recently discovered that some SSL clients, including OpenSSL, accept weak RSA keys—known as export-grade keys—without asking for those I Export-grade refers to 512-bit RSA keys, the key strength that was approved by

Bruno Blanchet (Inria)
Introduction to cryptographic protocols
Year 2016-17