On the Practical Exploitability of Dual EC DRBG in TLS Implementations

Stephen Checkoway, Matthew Fredrikson, Ruben Niederhagen,
Matthew Green, Tanja Lange, Thomas Ristenpart,
Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham

May 6, 2014

Random numbers are important

- Cryptography needs random numbers to generate long-term secret keys for encryption and signatures.
- Many schemes expect random (or pseudorandom) numbers, e.g.
  - ephemeral keys for DH key exchange,
  - nonces for digital signatures,
  - nonces in authenticated encryption.
- Nonce reuse can reveal long-term secret keys (e.g. PlayStation disaster)
- DSA/ECDSA are so touchy that biased nonces are enough to break them.

Random numbers are important to the NSA

- Cryptography needs random numbers to generate long-term secret keys for encryption and signatures.
- Many schemes expect random (or pseudorandom) numbers, e.g.
  - ephemeral keys for DH key exchange,
  - nonces for digital signatures,
  - nonces in authenticated encryption.
- Nonce reuse can reveal long-term secret keys (e.g. PlayStation disaster)
- DSA/ECDSA are so touchy that biased nonces are enough to break them.

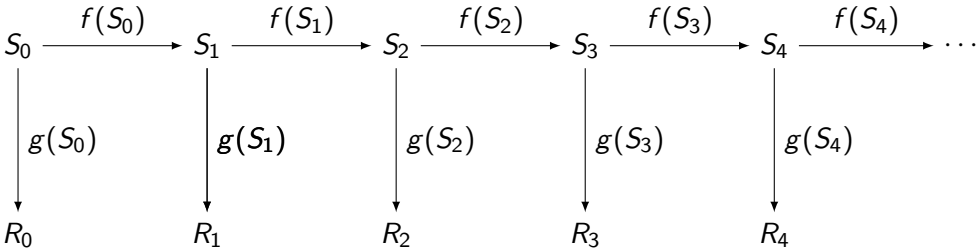
Snowden at SXSW:

> *[..] we know that these encryption algorithms we are using today work typically it is the random number generators that are attacked as opposed to the encryption algorithms themselves.*

Pseudo-random-number generators

Crypto libraries expand short seed into long stream of random bits.
Random bits are used as secret keys, DSA nonces, ...

The usual structure, starting from short seed S_1:



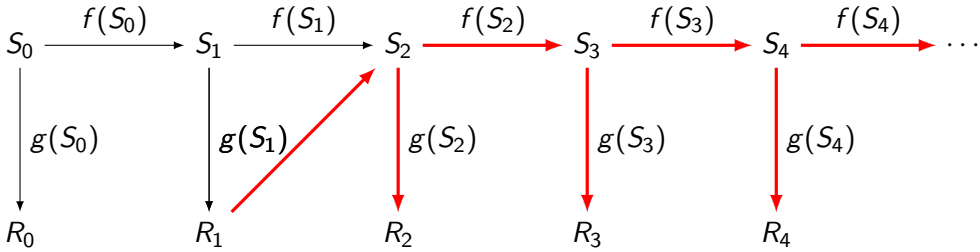
XXX's mission: Predict the "random" output bits.

1. Create protocols that directly output R_1 for some reason.

Pseudo-random-number generators

Crypto libraries expand short seed into long stream of random bits.
Random bits are used as secret keys, DSA nonces, ...

The usual structure, starting from short seed S_1:



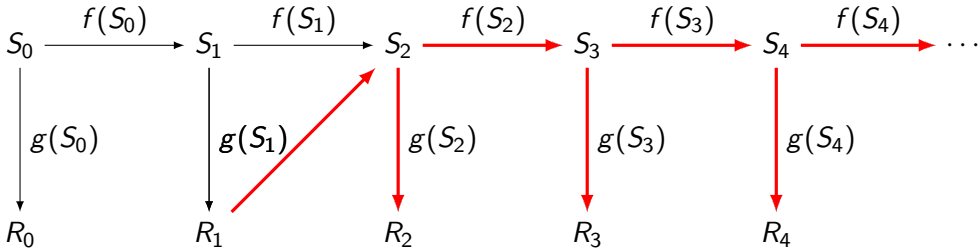
XXX's mission: Predict the "random" output bits.

1. Create protocols that directly output R_1 for some reason.
2. Design f, g with back door from R_n to S_{n+1}: i.e., get $f(S)$ from $g(S)$.

Pseudo-random-number generators

Crypto libraries expand short seed into long stream of random bits.
Random bits are used as secret keys, DSA nonces, ...

The usual structure, starting from short seed S_1:



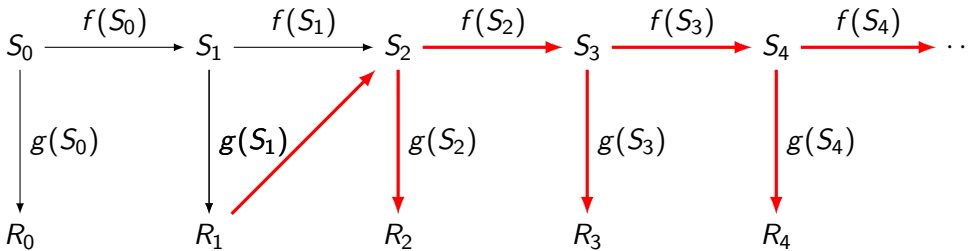
XXX's mission: Predict the "random" output bits.

1. Create protocols that directly output R_1 for some reason.
2. Design f, g with back door from R_n to S_{n+1}: i.e., get $f(S)$ from $g(S)$.
3. Standardize this design of f, g.

Pseudo-random-number generators

Crypto libraries expand short seed into long stream of random bits.
Random bits are used as secret keys, DSA nonces, ...

The usual structure, starting from short seed S_1:



XXX's mission: Predict the "random" output bits.

1. Create protocols that directly output R_1 for some reason.
2. Design f, g with back door from R_n to S_{n+1}: i.e., get $f(S)$ from $g(S)$.
3. Standardize this design of f, g.
4. Convince users to switch to this design: e.g., publish "security proof".

Elliptic curves

If P, Q are random points on a strong elliptic curve then it's hard to predict SP given SQ.

But if we know $P = kQ$ then it's easy: $SP = kSQ$.

Let's choose random Q, random k, define $P = kQ$. Standardize this P; Q; $f(S) = SP$; $g(S) = SQ$.

Elliptic curves

If P, Q are random points on a strong elliptic curve
then it's hard to predict SP given SQ.

But if we know $P = kQ$ then it's easy: $SP = kSQ$.

Let's choose random Q, random k, define $P = kQ$.
Standardize this P; Q; $f(S) = SP$; $g(S) = SQ$.

Wait a minute:
Curve points (x, y) don't look like random strings.
They satisfy public curve equation: $y^2 = x^3 - 3x + \text{constant}$.
This won't pass public review.

Elliptic curves

If P, Q are random points on a strong elliptic curve
then it's hard to predict SP given SQ.

But if we know $P = kQ$ then it's easy: $SP = kSQ$.

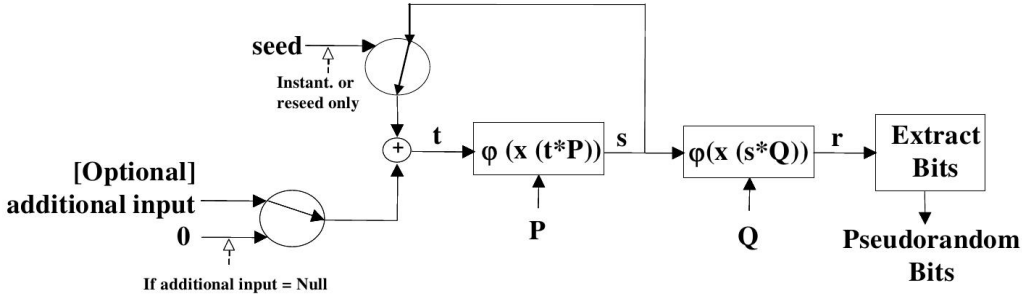
Let's choose random Q, random k, define $P = kQ$.
Standardize this P; Q; $f(S) = SP$; $g(S) = SQ$.

Wait a minute:
Curve points (x, y) don't look like random strings.
They satisfy public curve equation: $y^2 = x^3 - 3x + \text{constant}$.
This won't pass public review.

Solution: Let's throw away y and some bits of x.
Define $f(S) = \phi(x(SP))$, $g(S) = \phi(x(SQ))$ where ϕ omits 16 bits.
Not a big computation for us to recover SQ from $g(S)$.

DUAL_EC RNG: history part I

Earliest public source (?) June 2004, draft of ANSI X9.82:



φ gives all but the top 16 bits $\Rightarrow$ about 2^{15} points sQ match given string.

Claim:

Dual_EC_DRBG is based on the following hard problem, sometimes known as the "elliptic curve discrete logarithm problem" (ECDLP): given points P and Q on an elliptic curve of order n, find a such that $Q = aP$.

DUAL_EC RNG: common public history part II

Various public warning signals:

- Gjøsteen (March 2006): output sequence is biased.
  "While the practical impact of these results are modest, it is hard to see how these flaws would be acceptable in a pseudo-random bit generator based on symmetric cryptographic primitives. They should not be accepted in a generator based on number-theoretic assumptions."
- Brown (March 2006): security "proof"
  "This proof makes essential use of Q being random." If d with $dQ = P$ is known then $dR_i = S_{i+1}$, concludes that there might be distinguisher.
- Sidorenko & Schoenmakers (May 2006): output sequence is even more biased.
  Answer: Too late to change, already implemented.
- Shumow & Ferguson (August 2007): Backdoor if d is known.
- NIST SP800-90 gets appendix about choosing points verifiably at random, but requires use of standardized P, Q for FIPS-140 validation.

September 2013: NSA Bullrun program

- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.

September 2013: NSA Bullrun program

- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.

NYT:

> *the NSA had inserted a back door into a 2006 standard adopted by NIST [..] called the Dual EC DRBG standard.*

September 2013: NSA Bullrun program

- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.

NYT:

> *the NSA had inserted a back door into a 2006 standard adopted by NIST [..] called the Dual EC DRBG standard.*

. . . but surely nobody uses that!?!

September 2013: NSA Bullrun program

- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.

NYT:

> *the NSA had inserted a back door into a 2006 standard adopted by NIST [..] called the Dual EC DRBG standard.*

. . . but surely nobody uses that!?!

NIST's DRBG Validation List: more than 70 validations of Dual_EC_DRBG;
RSA's BSAFE has Dual_EC_DRBG enabled as default,.

September 2013: NSA Bullrun program

- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.

NYT:

> *the NSA had inserted a back door into a 2006 standard adopted by NIST [..] called the Dual EC DRBG standard.*

. . . but surely nobody uses that!?!

NIST's DRBG Validation List: more than 70 validations of Dual_EC_DRBG;
RSA's BSAFE has Dual_EC_DRBG enabled as default,.

NIST re-opens discussions on SP800.90; recommmends against using Dual_EC.
RSA suggests changing default in BSAFE.

21 April 2014 NIST removes Dual EC from the standard.

How expensive is using the backdoor?

Rereading the standard:

" $x(A)$ is the x-coordinate of the point A on the curve, given in affine coordinates. An implementation may choose to represent points internally using other coordinate systems; for instance, when efficiency is a primary concern. In this case, a point shall be translated back to affine coordinates before $x()$ is applied."

How expensive is using the backdoor?

Rereading the standard:

" $x(A)$ is the x-coordinate of the point A on the curve, given in affine coordinates. An implementation may choose to represent points internally using other coordinate systems; for instance, when efficiency is a primary concern. In this case, a point shall be translated back to affine coordinates before $x()$ is applied."

Given $r_i = \varphi(x(s_i Q))$, $r_{i+1} = \varphi(x(s_{i+1} Q))$, and NSA backdoor $d = log_P(Q)$.

1. Expand r_i to candidate $Q_i = s_i Q$, [50% chance; if fail move on to next candidate]
2. compute candidate $P_{i+1} = dQ_i$ and candidate $s_{i+1} = \varphi(x(P_{i+1}))$
3. check, $\varphi(x(s_{i+1} Q))$ against r_{i+1}. [if fail, goto 1.; else most likely done!]

How expensive is using the backdoor?

Rereading the standard:
" $x(A)$ is the x-coordinate of the point A on the curve, given in affine coordinates. An implementation may choose to represent points internally using other coordinate systems; for instance, when efficiency is a primary concern. In this case, a point shall be translated back to affine coordinates before $x()$ is applied."

Given $r_i = \varphi(x(s_i Q))$, $r_{i+1} = \varphi(x(s_{i+1} Q))$, and NSA backdoor $d = log_P(Q)$.

1. Expand r_i to candidate $Q_i = s_i Q$, [50% chance; if fail move on to next candidate]
2. compute candidate $P_{i+1} = dQ_i$ and candidate $s_{i+1} = \varphi(x(P_{i+1}))$
3. check, $\varphi(x(s_{i+1} Q))$ against r_{i+1}. [if fail, goto 1.; else most likely done!]

Initial timings on i7 M620 Core

missing	16 bits	24 bits	32 bits
1 core	20s	85m	15d4h

How expensive is using the backdoor?

Rereading the standard:
" $x(A)$ is the x-coordinate of the point A on the curve, given in affine coordinates. An implementation may choose to represent points internally using other coordinate systems; for instance, when efficiency is a primary concern. In this case, a point shall be translated back to affine coordinates before $x()$ is applied."

Given $r_i = \varphi(x(s_i Q))$, $r_{i+1} = \varphi(x(s_{i+1} Q))$, and NSA backdoor $d = log_P(Q)$.

1. Expand r_i to candidate $Q_i = s_i Q$, [50% chance; if fail move on to next candidate]
2. compute candidate $P_{i+1} = dQ_i$ and candidate $s_{i+1} = \varphi(x(P_{i+1}))$
3. check, $\varphi(x(s_{i+1} Q))$ against r_{i+1}. [if fail, goto 1.; else most likely done!]

Initial timings on i7 M620 Core

missing	16 bits	24 bits	32 bits
1 core	20s	85m	15d4h
64k cores			20s

How expensive is using the backdoor?

Rereading the standard:

" $x(A)$ is the x-coordinate of the point A on the curve, given in affine coordinates. An implementation may choose to represent points internally using other coordinate systems; for instance, when efficiency is a primary concern. In this case, a point shall be translated back to affine coordinates before $x()$ is applied."

Given $r_i = \varphi(x(s_i Q))$, $r_{i+1} = \varphi(x(s_{i+1} Q))$, and NSA backdoor $d = log_P(Q)$.

1. Expand r_i to candidate $Q_i = s_i Q$, [50% chance; if fail move on to next candidate]
2. compute candidate $P_{i+1} = dQ_i$ and candidate $s_{i+1} = \varphi(x(P_{i+1}))$
3. check, $\varphi(x(s_{i+1} Q))$ against r_{i+1}. [if fail, goto 1.; else most likely done!]

Initial timings on i7 M620 Core

missing	16 bits	24 bits	32 bits
1 core	20s	85m	15d4h
64k cores			20s

From the standard:

"For performance reasons, the value of outlen should be set to the maximum value as provided in Table 4."

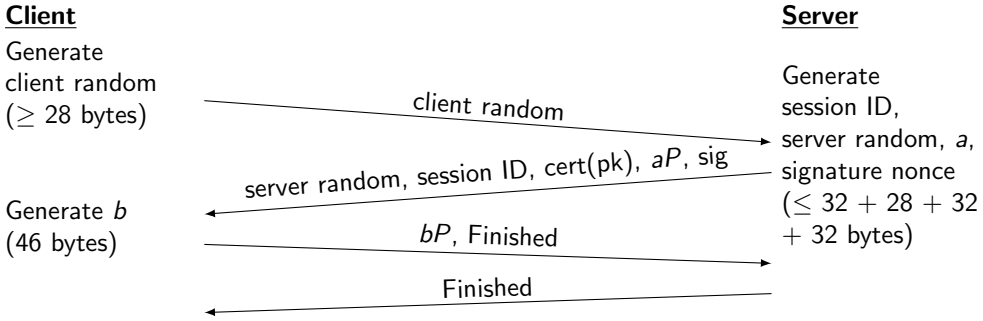
Don't give us fewer bits!

SSL/TLS/HTTPS

How are RNGs actually used? Do implementations actually leak enough of R_1?

SSL/TLS/HTTPS

How are RNGs actually used? Do implementations actually leak enough of R_1?



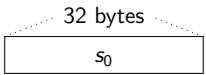
$\mathrm{MS} = \mathrm{PRF}(x(abP), \text{"master secret"}, \text{client random} \frown\frown \text{server random})$

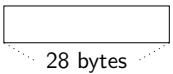
Dual EC in TLS

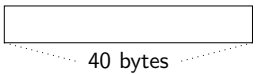
Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

Points Q and P on an elliptic curve.



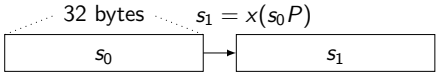




Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

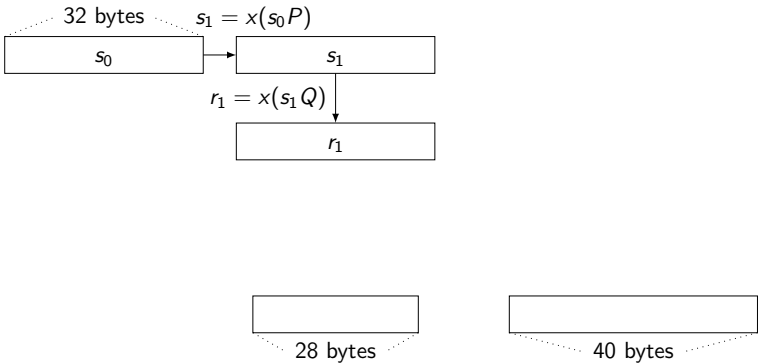
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

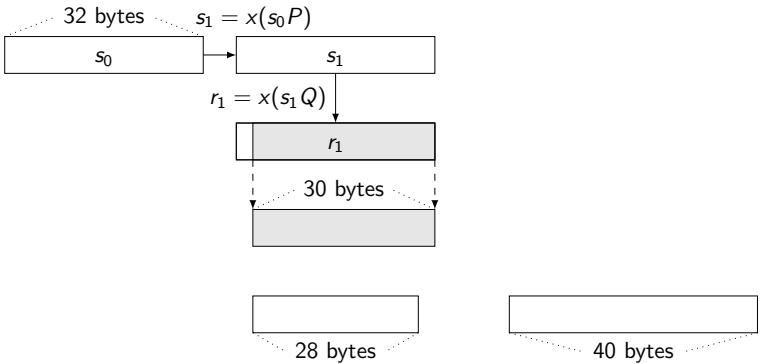
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

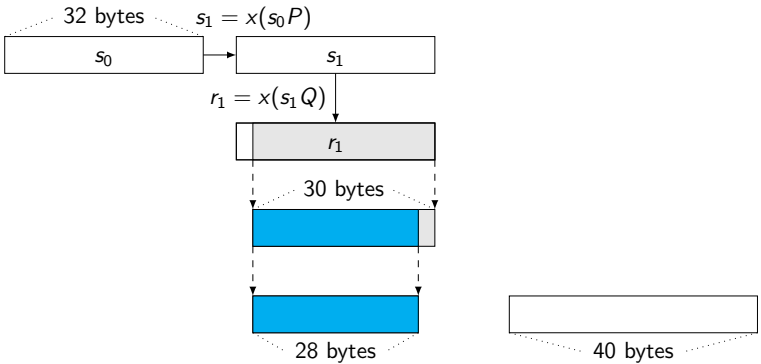
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

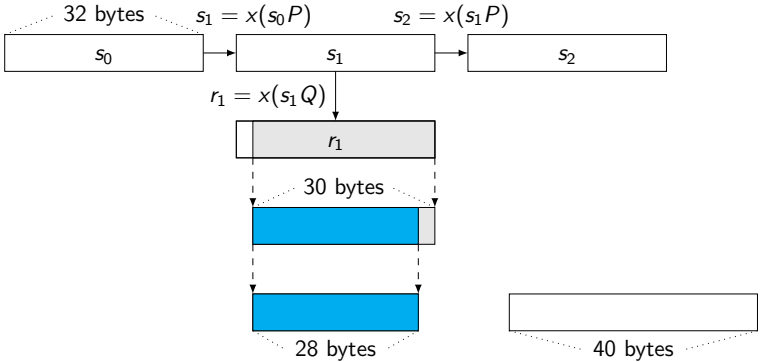
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

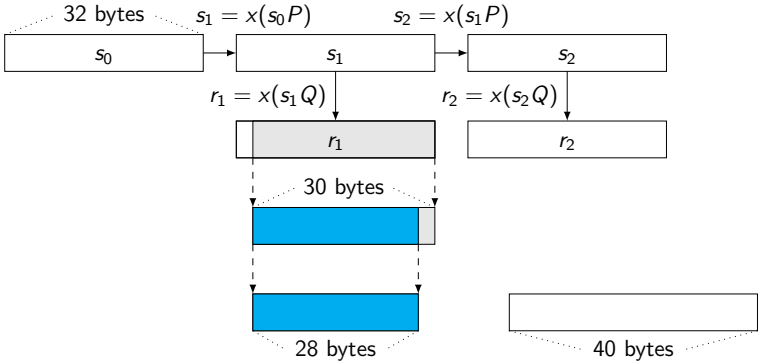
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

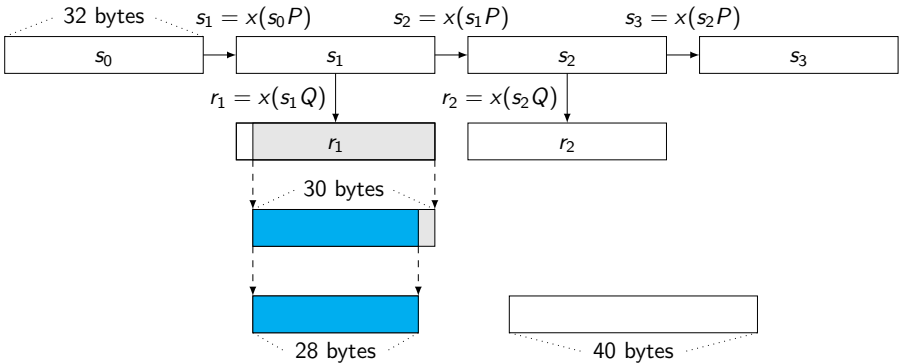
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

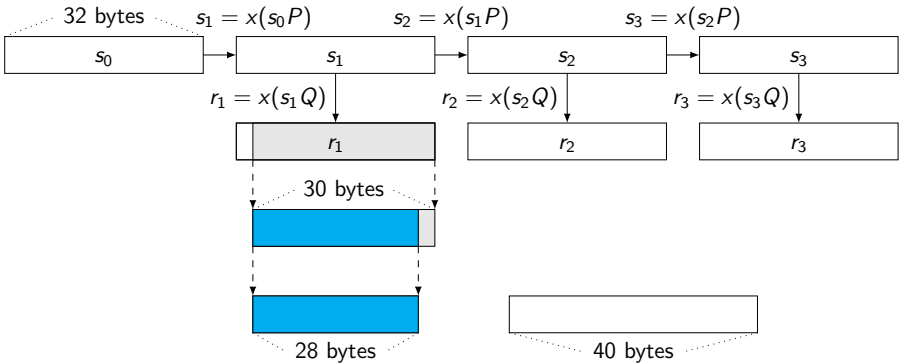
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

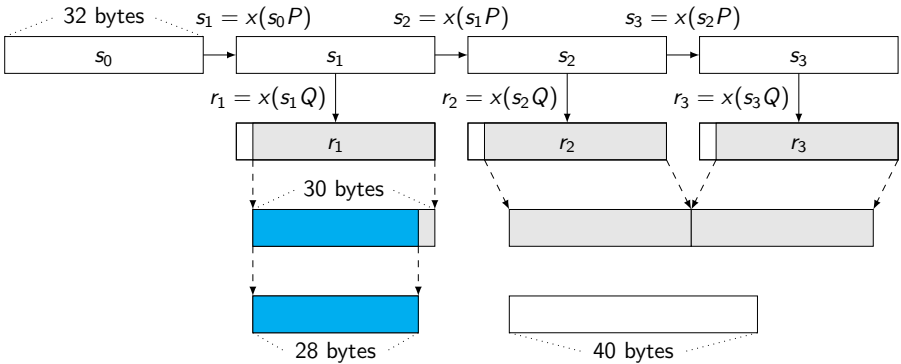
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

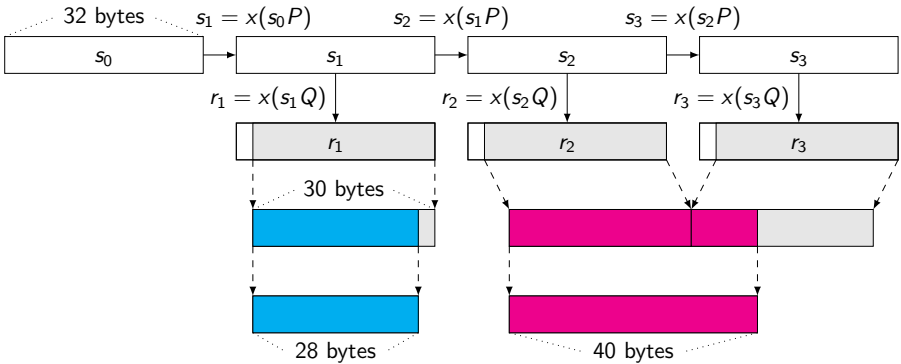
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

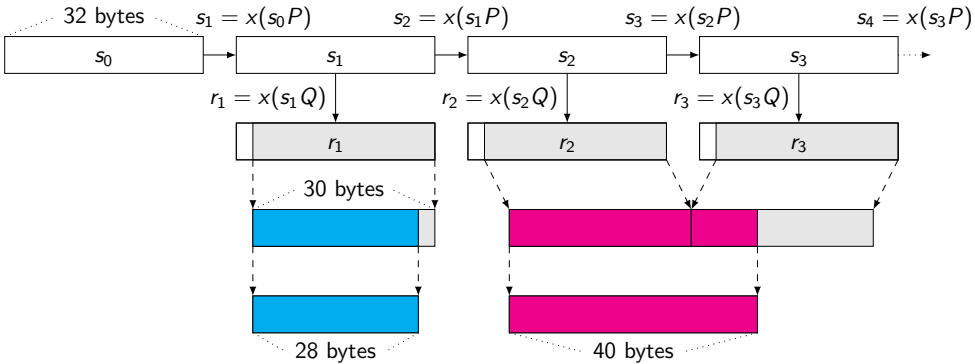
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

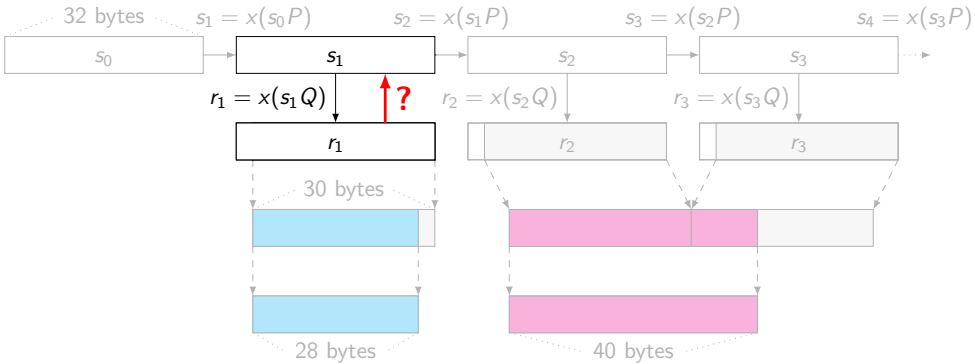
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

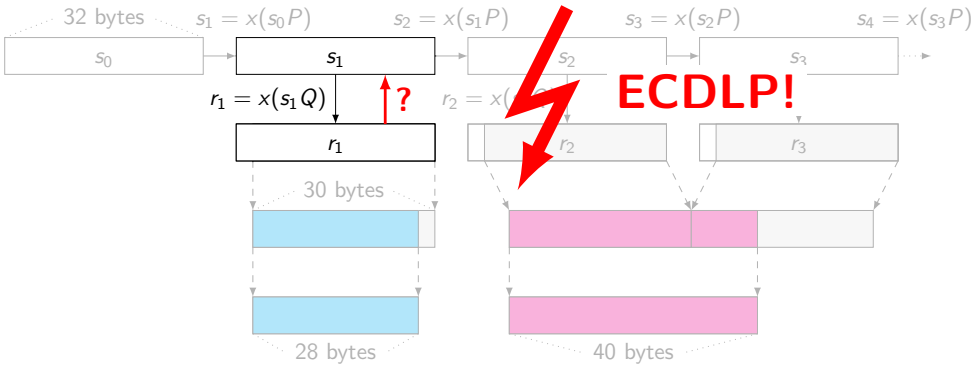
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Dual EC in TLS

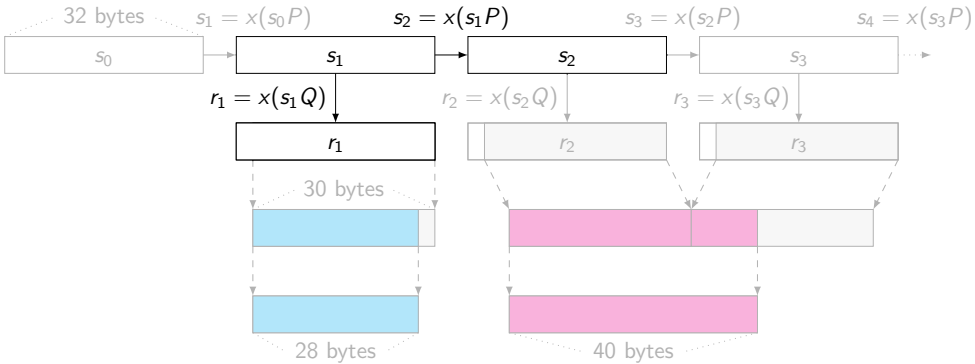
Points Q and P on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Basic attack

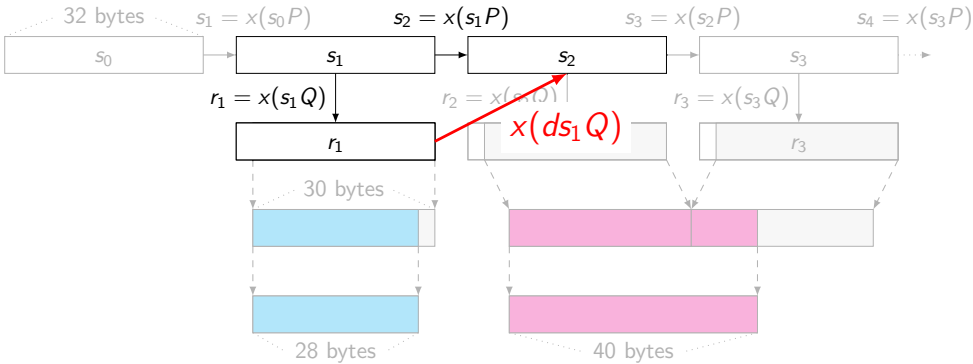
Points Q and $P = dQ$ on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Basic attack

Points Q and $P = dQ$ on an elliptic curve.

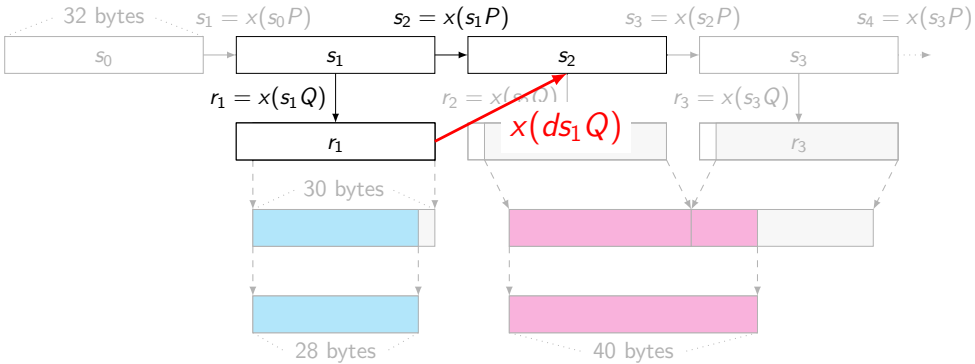


Graphic thanks to Ruben Niederhagen.

Basic attack

Points Q and $P = dQ$ on an elliptic curve.

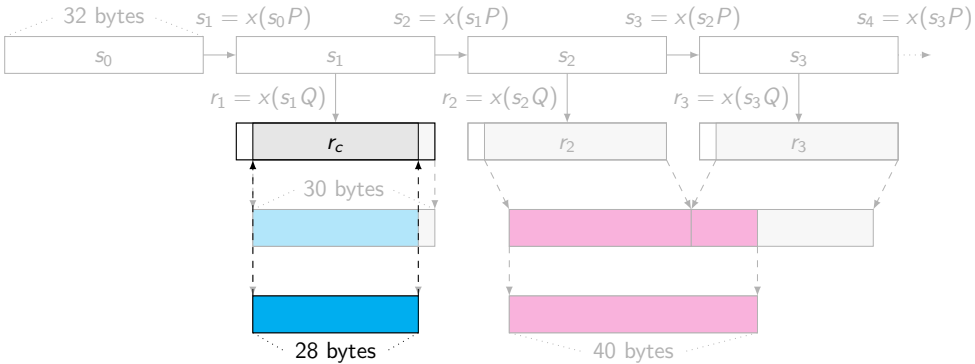
$$s_2 = x(s_1 P) = x(s_1 dQ)$$



Graphic thanks to Ruben Niederhagen.

Basic attack

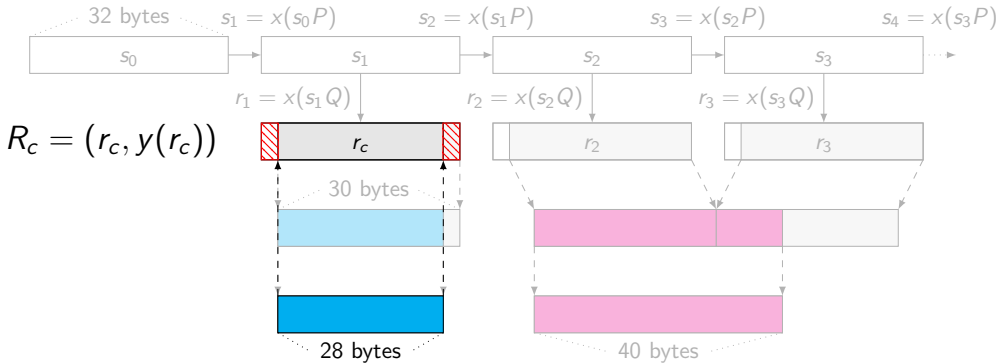
Points Q and $P = dQ$ on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Basic attack

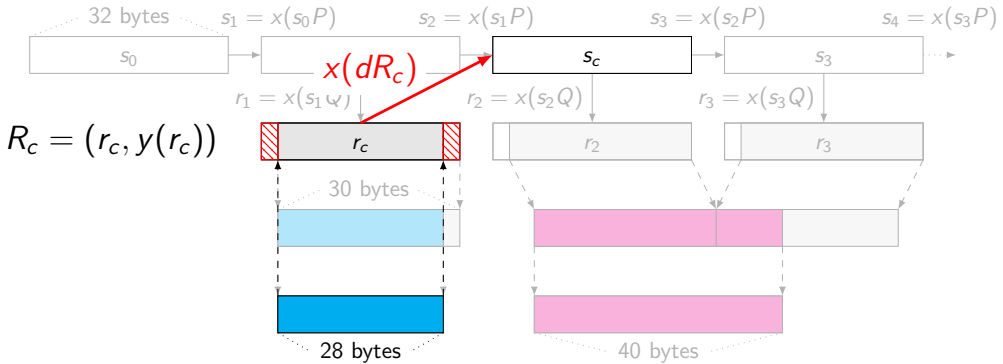
Points Q and $P = dQ$ on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

Basic attack

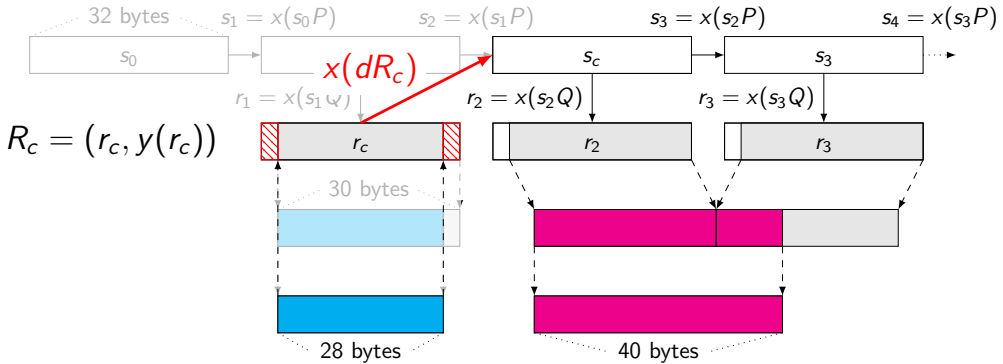
Points Q and $P = dQ$ on an elliptic curve.



Graphic thanks to Ruben Niederhagen.

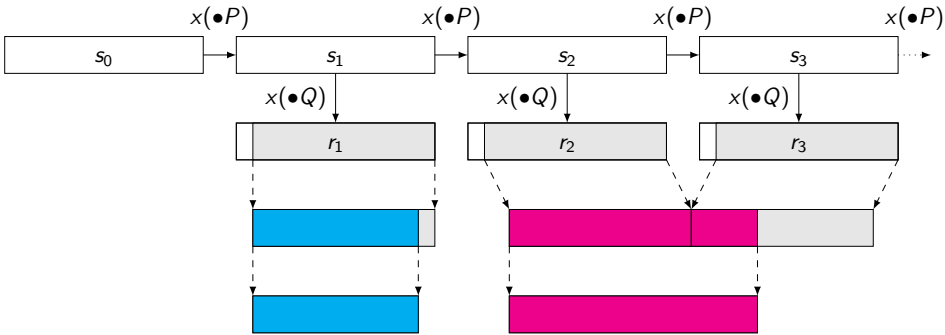
Basic attack

Points Q and $P = dQ$ on an elliptic curve.



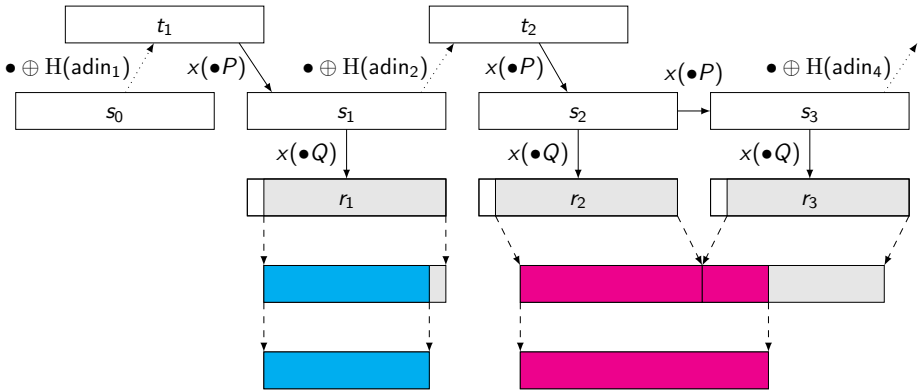
Graphic thanks to Ruben Niederhagen.

Dual EC in TLS



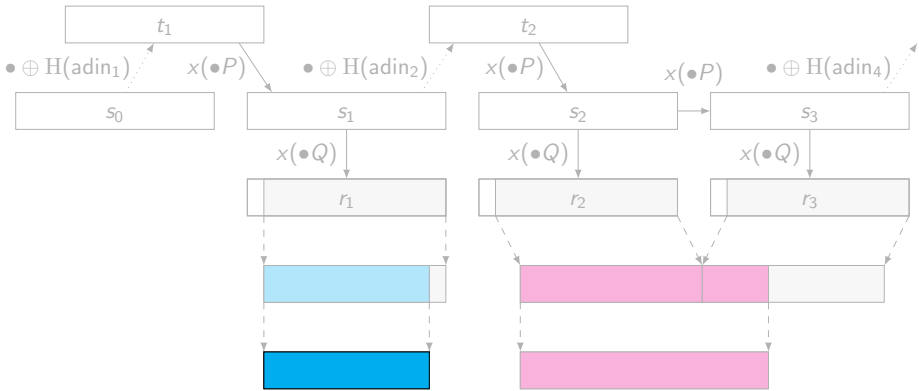
Graphic thanks to Ruben Niederhagen.

NIST SP800-90 in June 2006



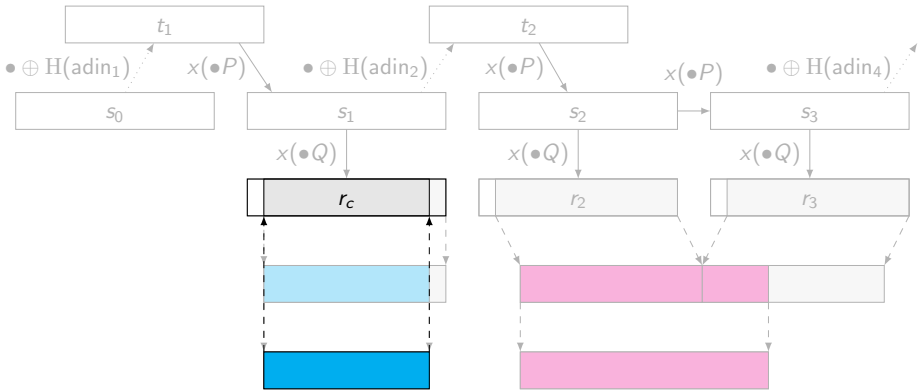
Graphic thanks to Ruben Niederhagen.

NIST SP800-90 in June 2006



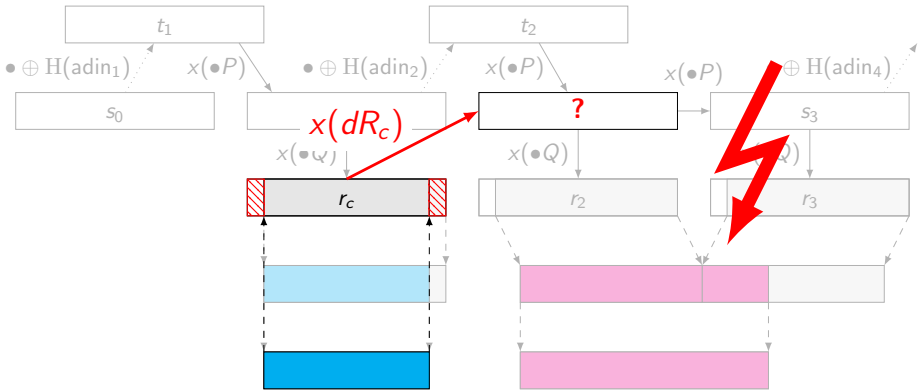
Graphic thanks to Ruben Niederhagen.

NIST SP800-90 in June 2006



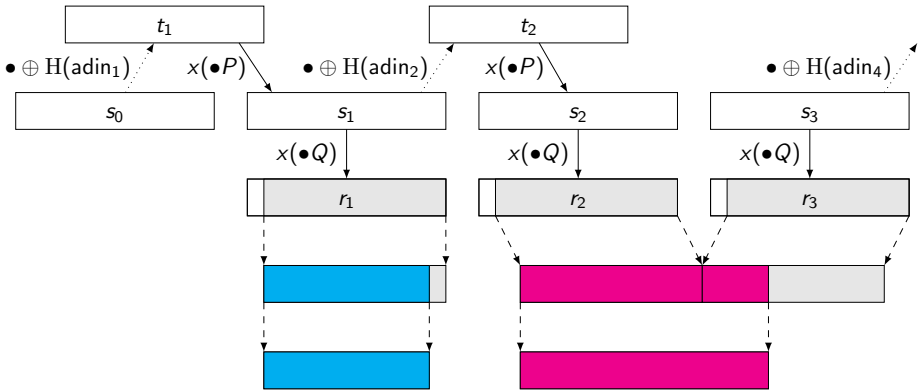
Graphic thanks to Ruben Niederhagen.

NIST SP800-90 in June 2006



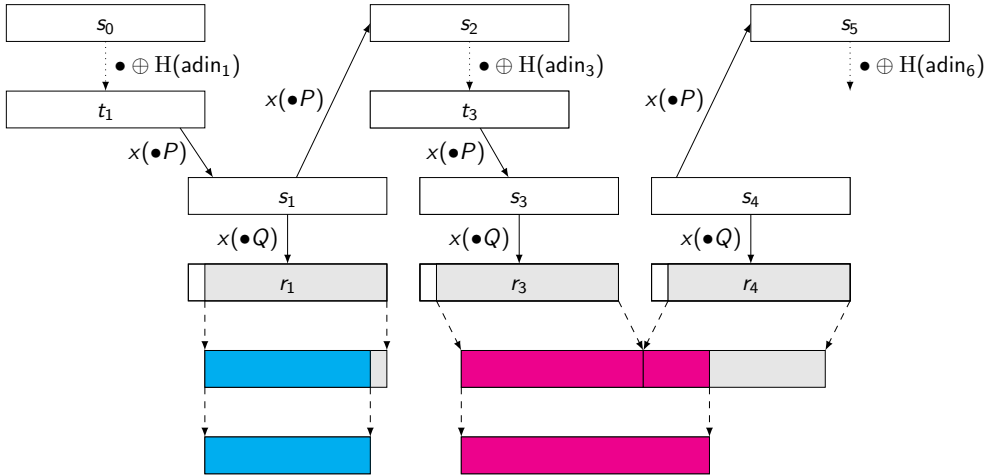
Graphic thanks to Ruben Niederhagen.

Dual EC in TLS



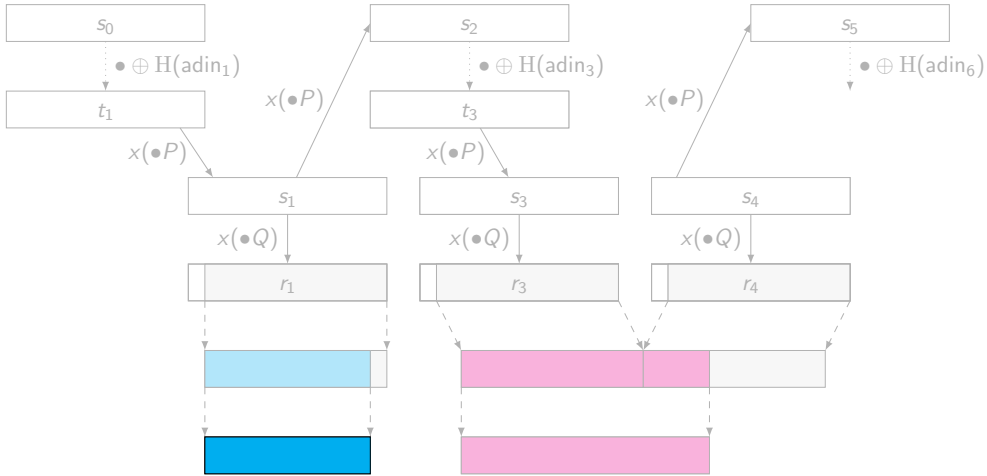
Graphic thanks to Ruben Niederhagen.

NIST SP800-90 in March 2007



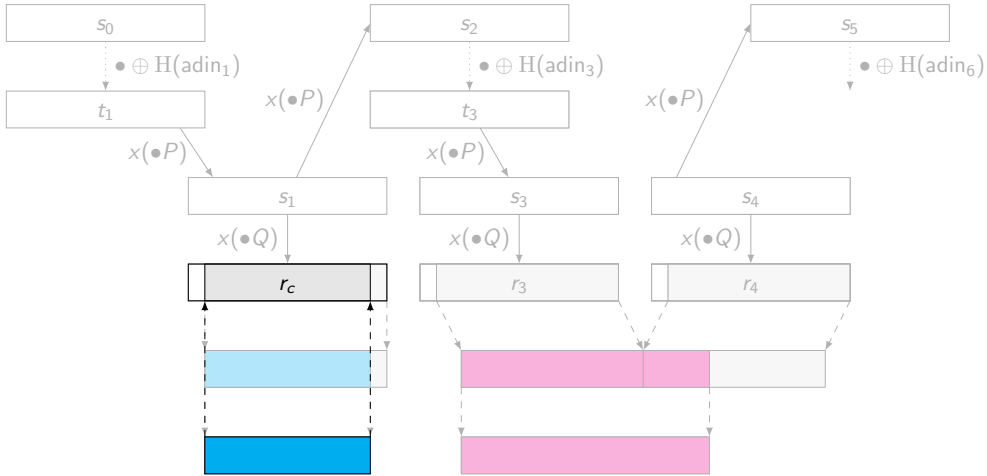
Graphic thanks to Ruben Niederhagen.

NIST SP800-90 in March 2007



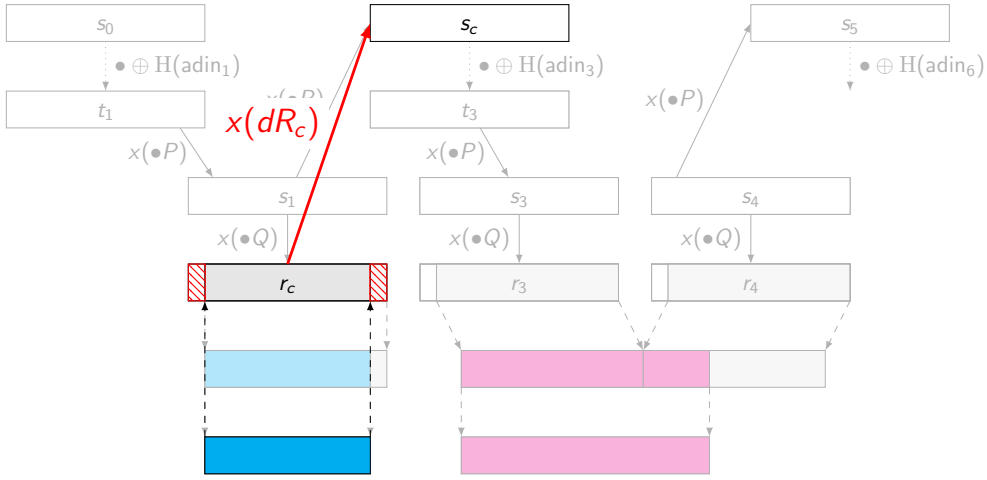
Graphic thanks to Ruben Niederhagen.

NIST SP800-90 in March 2007



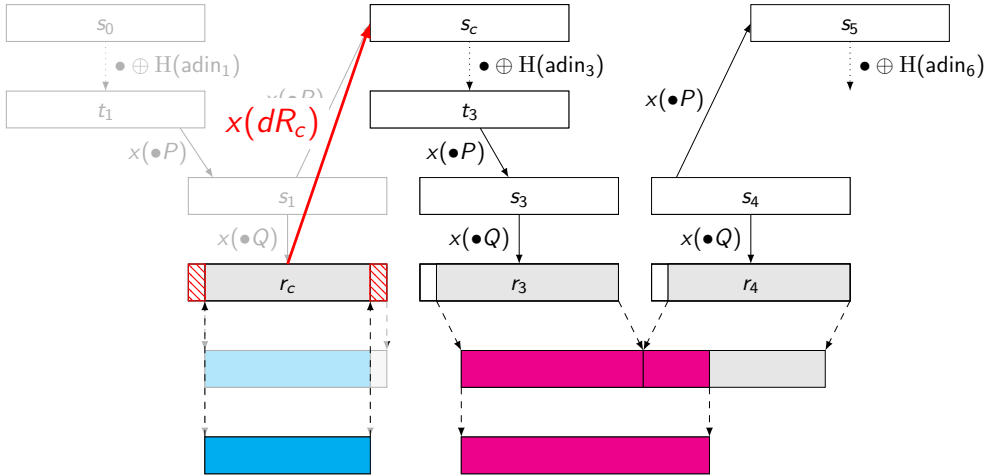
Graphic thanks to Ruben Niederhagen.

NIST SP800-90 in March 2007



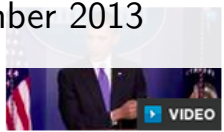
Graphic thanks to Ruben Niederhagen.

NIST SP800-90 in March 2007



Graphic thanks to Ruben Niederhagen.

December 2013



Obama on surveillance: "There may be another way of skinning the cat"

(Reuters) - As a key part of a campaign to embed encryption software that it could crack into widely used computer products, the U.S. National Security Agency arranged a secret $10 million contract with RSA, one of the most influential firms in the computer security industry, Reuters has learned.

Documents leaked by former NSA contractor Edward Snowden show that the NSA created and promulgated a flawed formula for generating random numbers to create a "back door" in encryption products, the New York Times reported in September. Reuters later reported that RSA became the most important distributor of that formula by rolling it into a software tool called Bsafe that is used to enhance security in personal computers and many other products.

Undisclosed until now was that RSA received $10 million in a deal that set the NSA formula as the preferred, or default, method for number generation in the BSafe software, according to two sources familiar with the contract. Although that sum might seem paltry, it represented more than a third of the revenue that the relevant division at RSA had taken in during the entire previous year, securities filings show.

December 22, 2013

Recent press coverage has asserted that RSA entered into a "secret contract" with the NSA to incorporate a known flawed random number generator into its BSAFE encryption libraries. We categorically deny this allegation.

We have worked with the NSA, both as a vendor and an active member of the security community. We have never kept this relationship a secret and in fact have openly publicized it. Our explicit goal has always been to strengthen commercial and government security.

Key points about our use of Dual EC DRBG in BSAFE are as follows:

- We made the decision to use Dual EC DRBG as the default in BSAFE toolkits in 2004, in the context of an industry-wide effort to develop newer, stronger methods of encryption. At that time, the NSA had a trusted role in the community-wide effort to strengthen, not weaken, encryption.
- This algorithm is only one of multiple choices available within BSAFE toolkits, and users have always been free to choose whichever one best suits their needs.
- We continued using the algorithm as an option within BSAFE toolkits as it gained acceptance as a NIST standard and because of its value in FIPS compliance. When concern surfaced around the algorithm in 2007, we continued to rely upon NIST as the arbiter of that discussion.

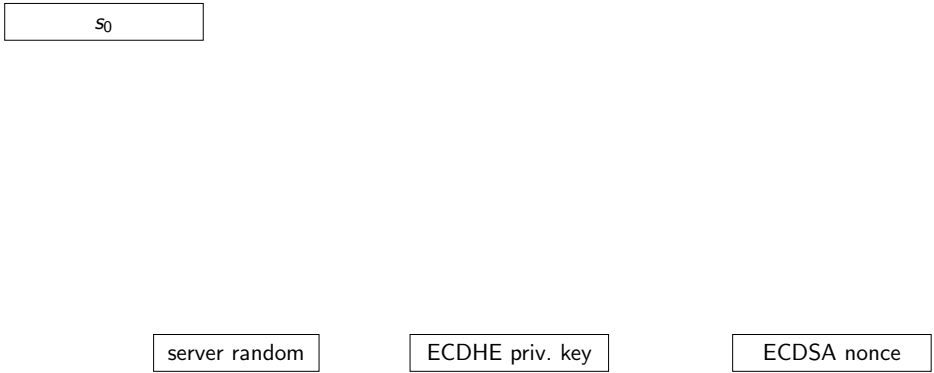
Attack – Example: BSAFE-Java

server random

ECDHE priv. key

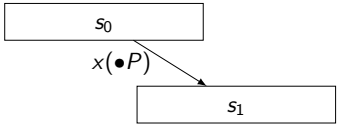
ECDSA nonce

Attack – Example: BSAFE-Java



Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



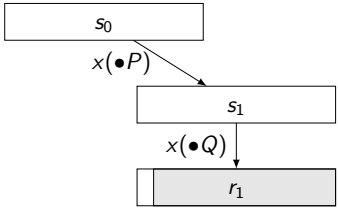
server random

ECDHE priv. key

ECDSA nonce

Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



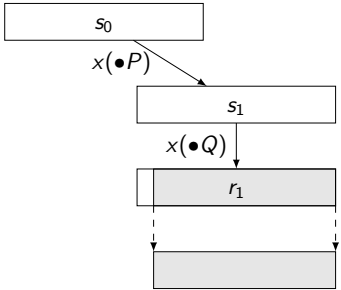
server random

ECDHE priv. key

ECDSA nonce

Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



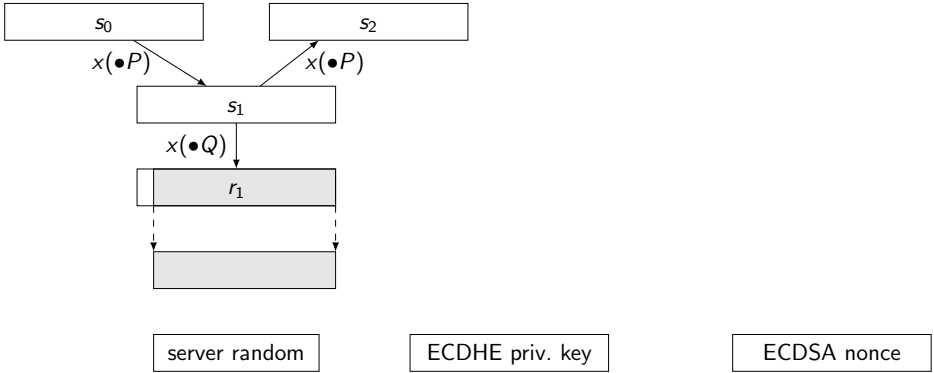
server random

ECDHE priv. key

ECDSA nonce

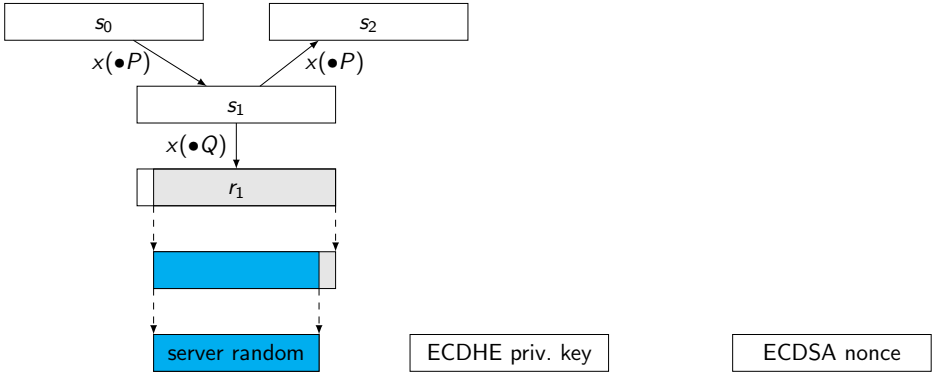
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



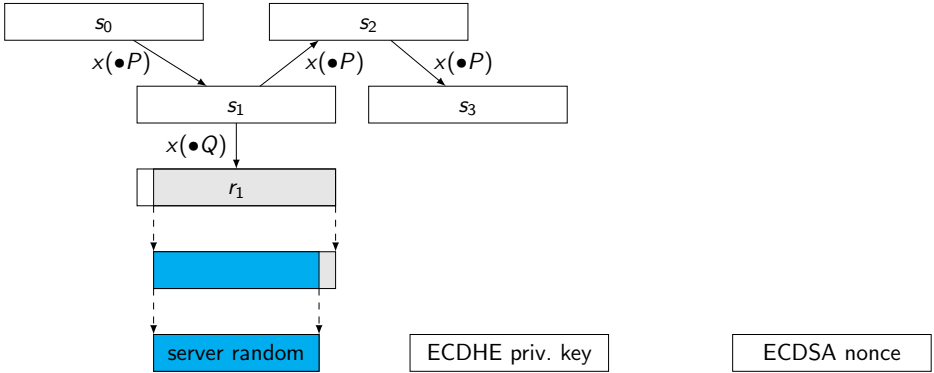
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



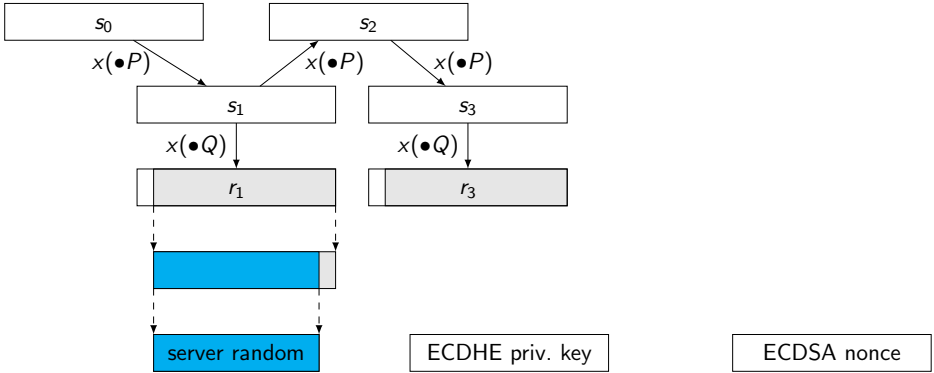
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



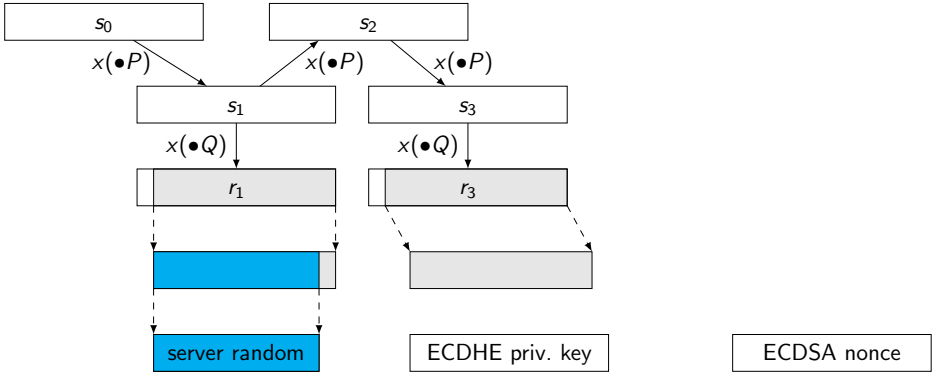
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



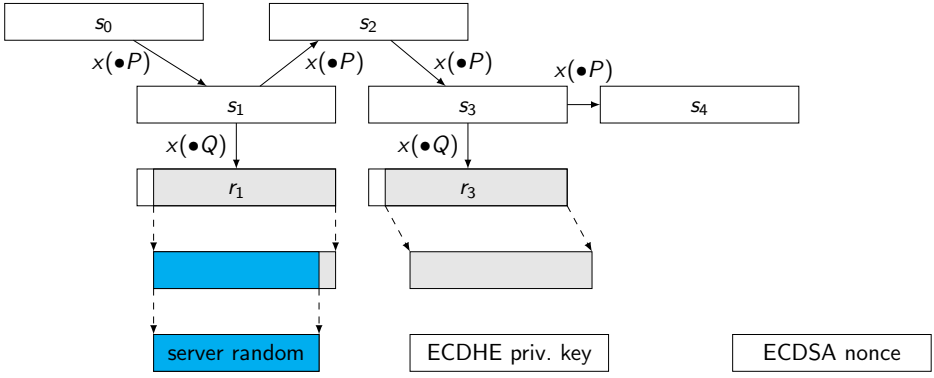
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



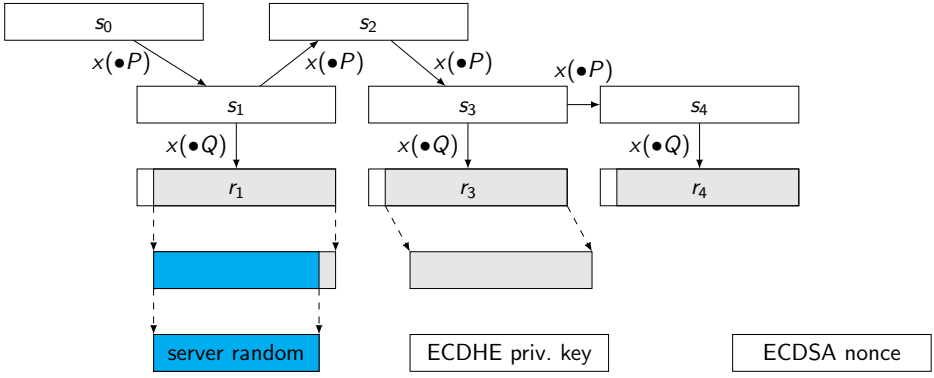
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



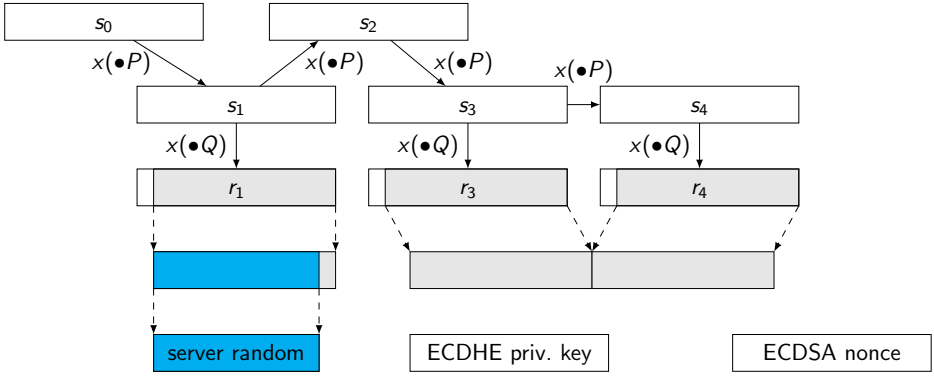
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



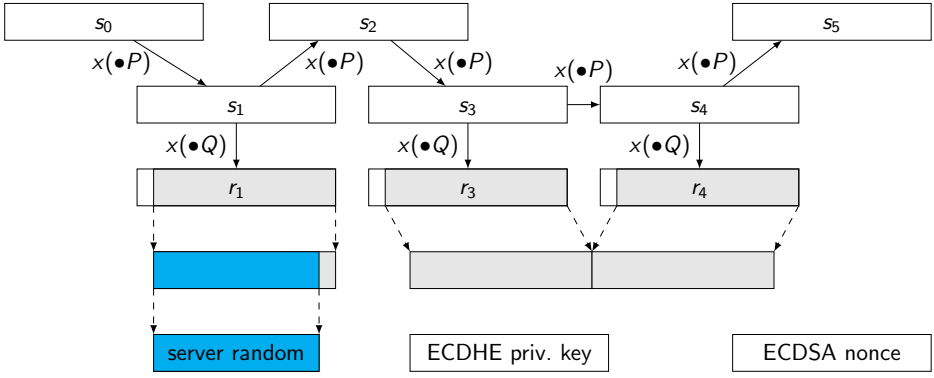
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



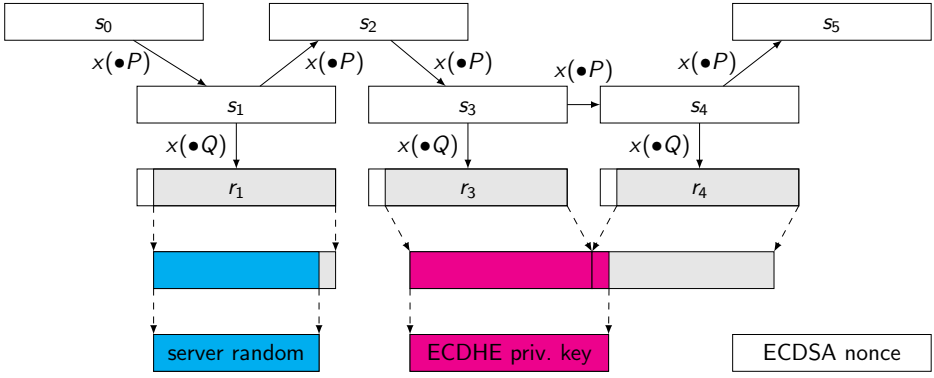
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



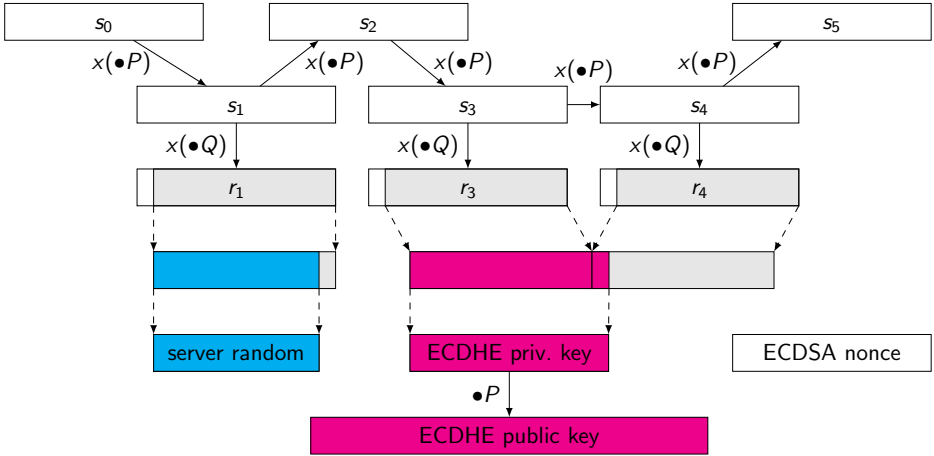
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



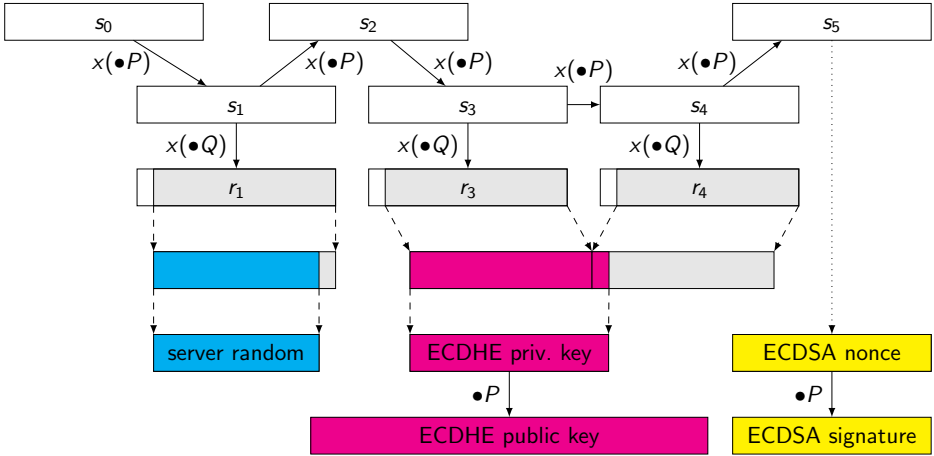
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



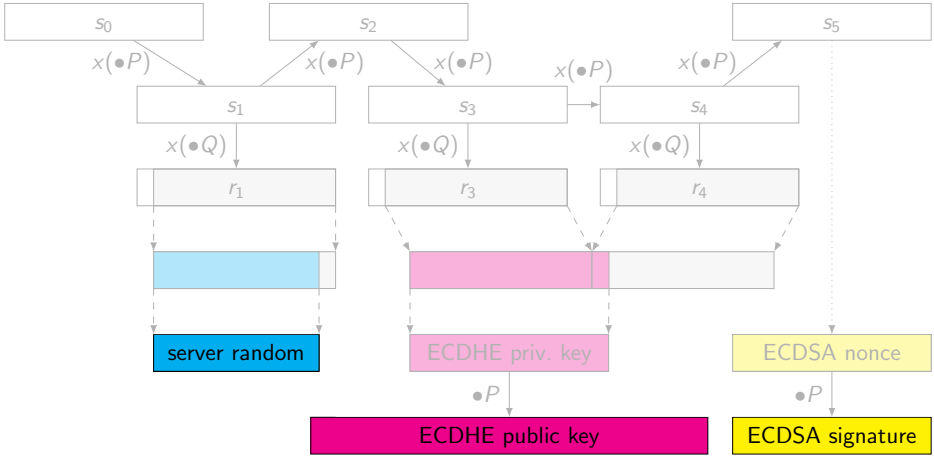
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



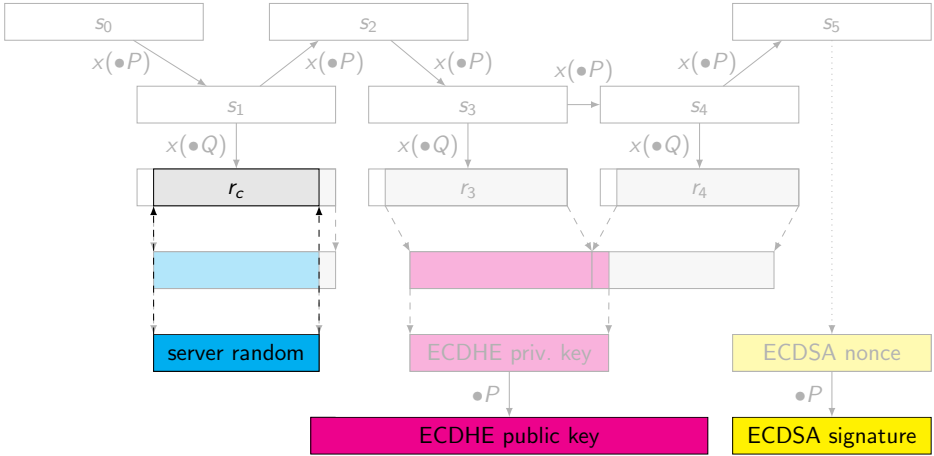
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



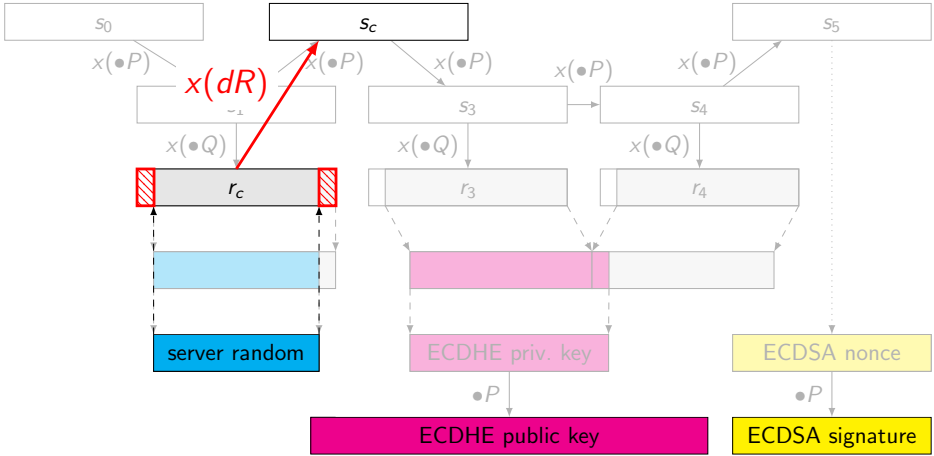
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



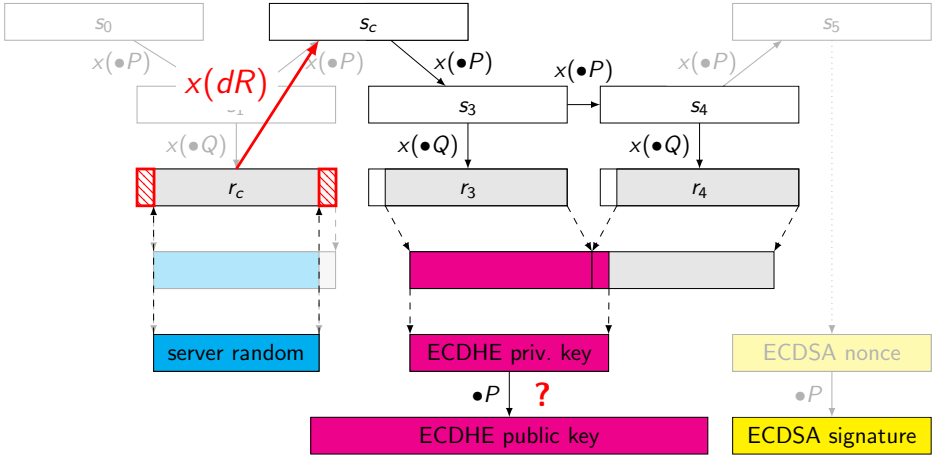
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



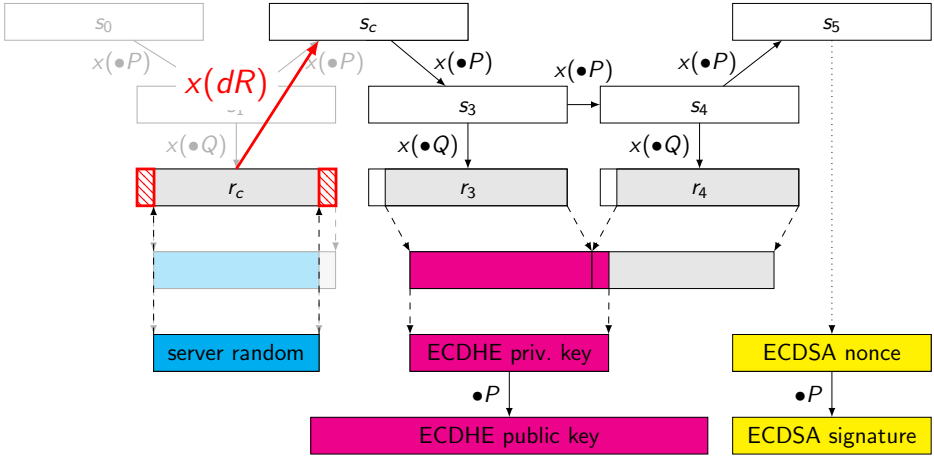
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



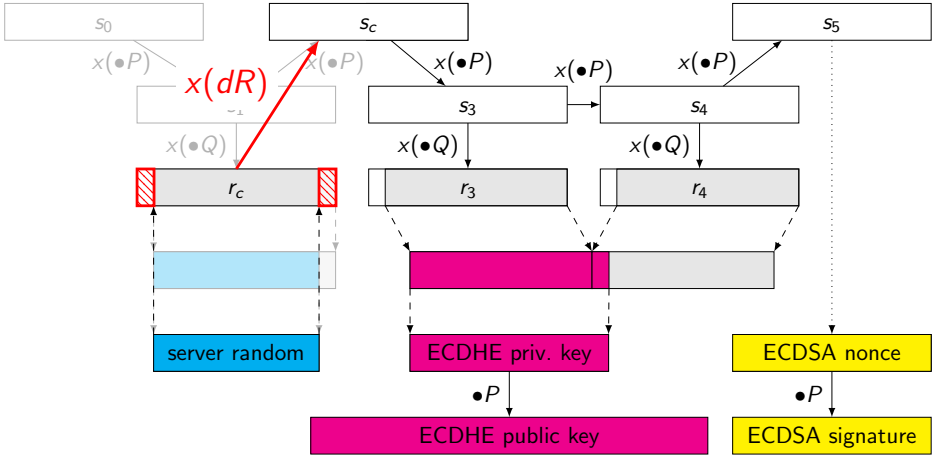
Graphics: Ruben Niederhagen.

Attack – Example: BSAFE-Java



Graphics: Ruben Niederhagen.

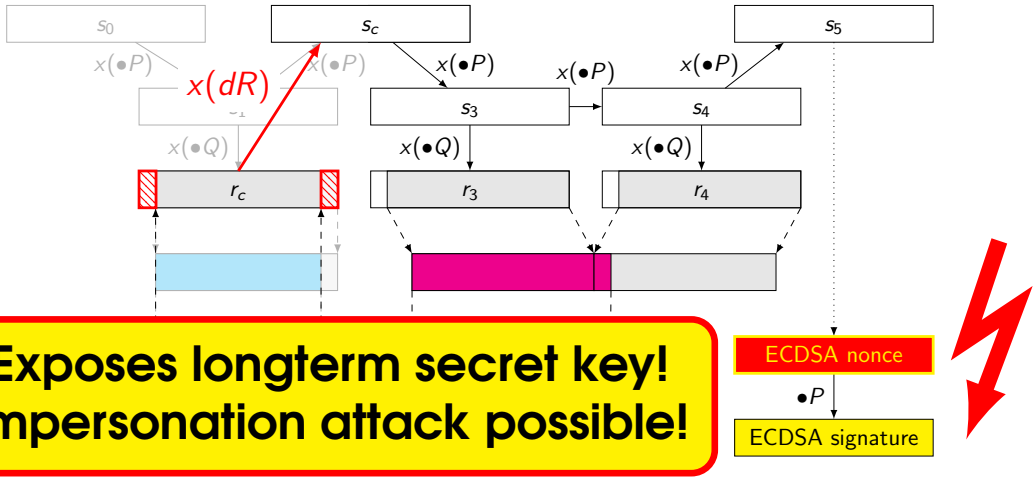
Attack – Example: BSAFE-Java



Graphics: Ruben Niederhagen.

average cost: $2^{31}(C_v + 5C_f)$

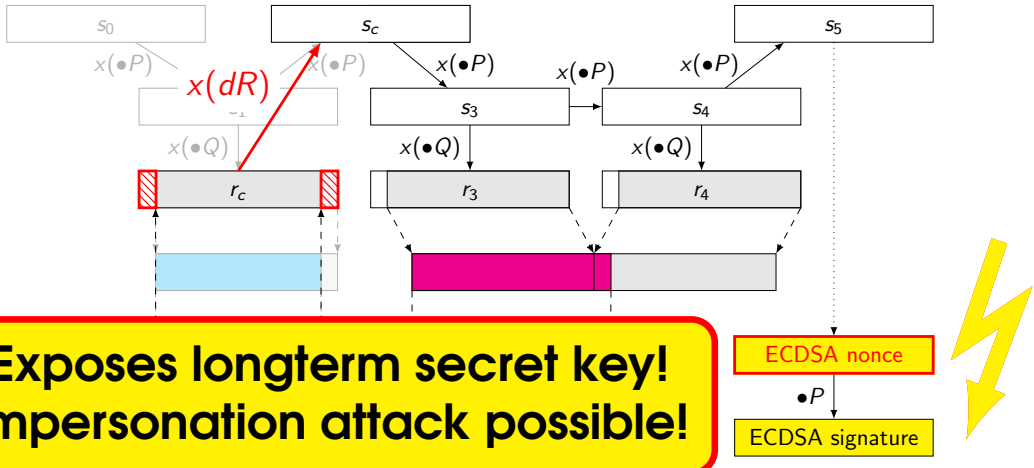
Attack – Example: BSAFE-Java



Graphics: Ruben Niederhagen.

average cost: $2^{31}(C_v + 5C_f)$

Attack – Example: BSAFE-Java



Graphics: Ruben Niederhagen.

average cost: $2^{31}(C_v + 5C_f)$

Some more fun with RSA's BSAFE-Java

No additional input,

Some more fun with RSA's BSAFE-Java

No additional input, **explicit watermark** in handshake $\Rightarrow$ easy recognition.

Some more fun with RSA's BSAFE-Java

No additional input, **explicit watermark** in handshake ⇒ easy recognition.

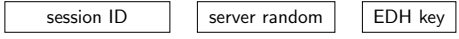
Alas, BSAFE does not give fresh randomness in session ID, so attack costs roughly 2^{32}.

```
Network Working Group                                  E. Rescorla
Internet-Draft                                          RTFM, Inc.
Intended status:  Informational                          M. Salter
Expires:  September 3, 2009              National Security Agency
                                                    March 02, 2009


                    Extended Random Values for TLS
                draft-rescorla-tls-extended-random-02.txt
```

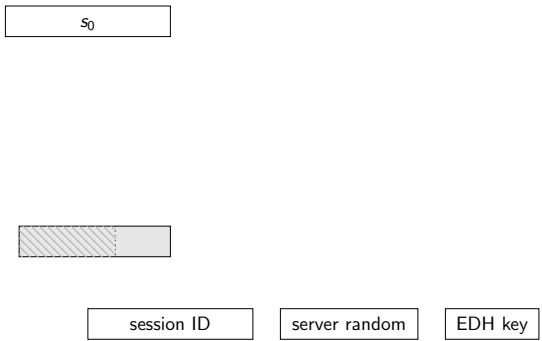
```
[..]  The rationale for this as stated by DoD is that the public
randomness for each side should be at least twice as long as the
security level for cryptographic parity, which makes the 224 bits of
randomness provided by the current TLS random values insufficient.
```

Attack – Example: BSAFE-C



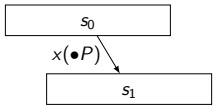
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



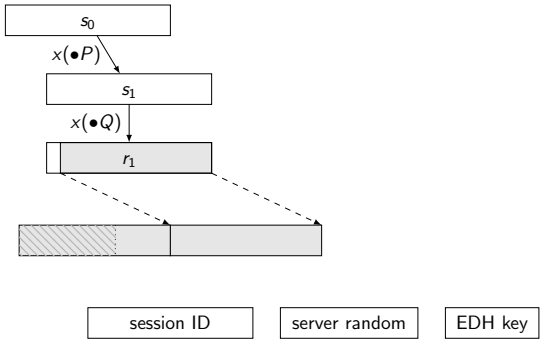
session ID | server random | EDH key

Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C

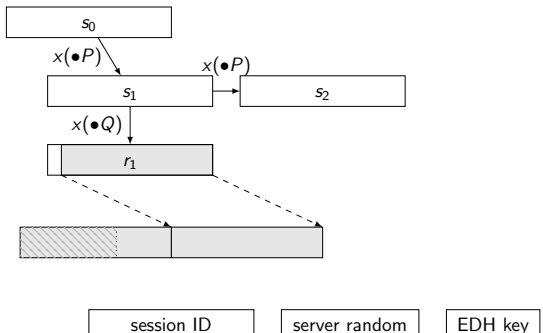
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



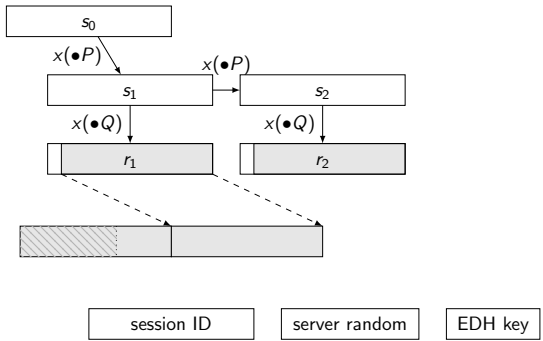
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



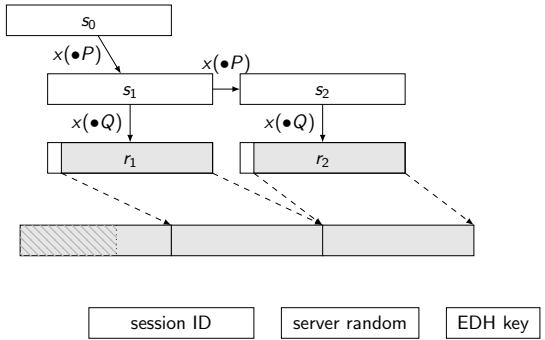
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



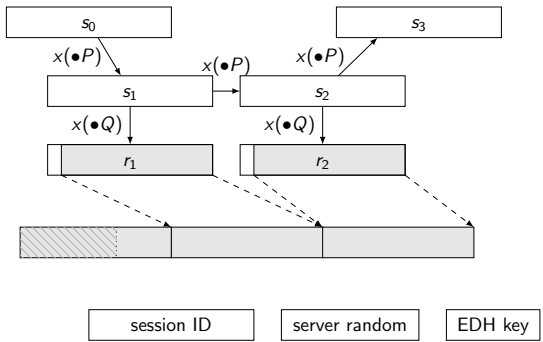
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



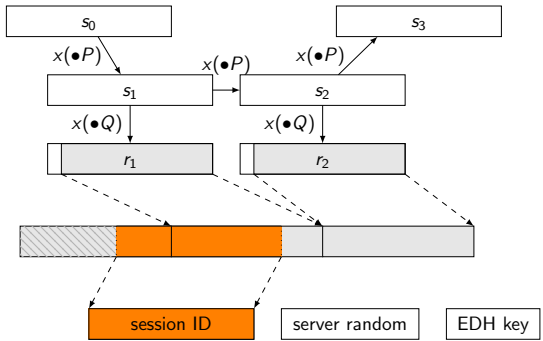
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



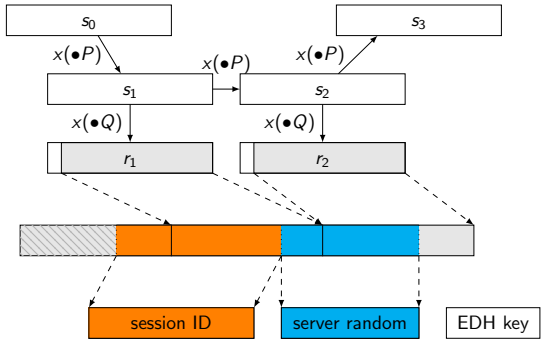
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



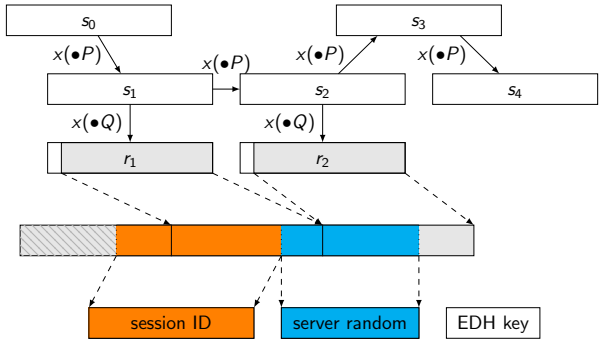
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



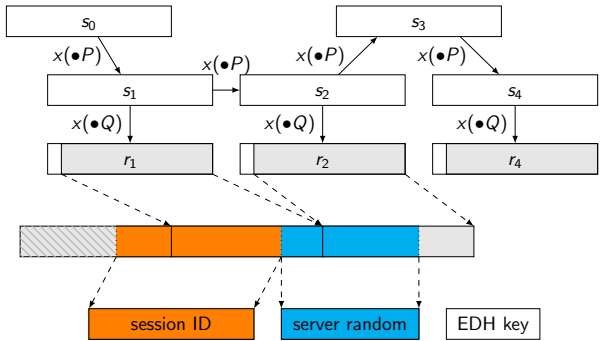
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



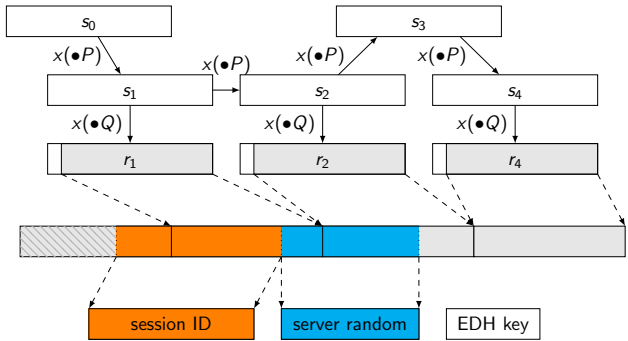
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



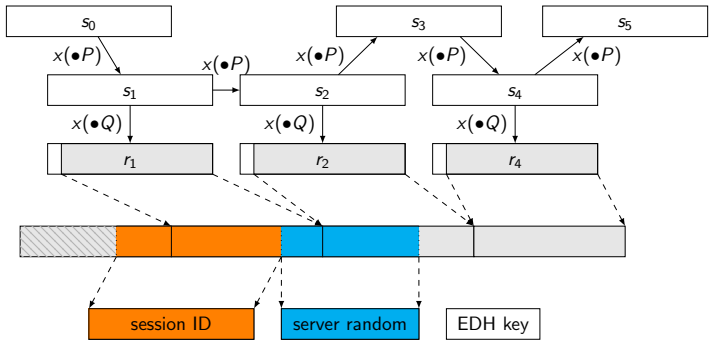
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



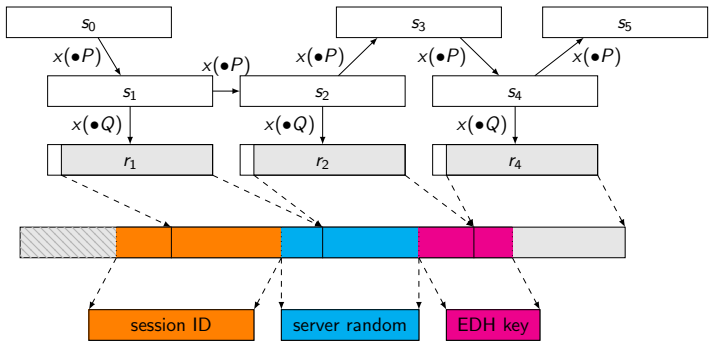
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



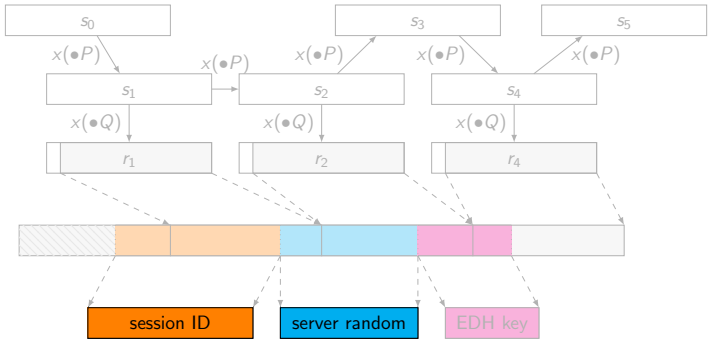
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



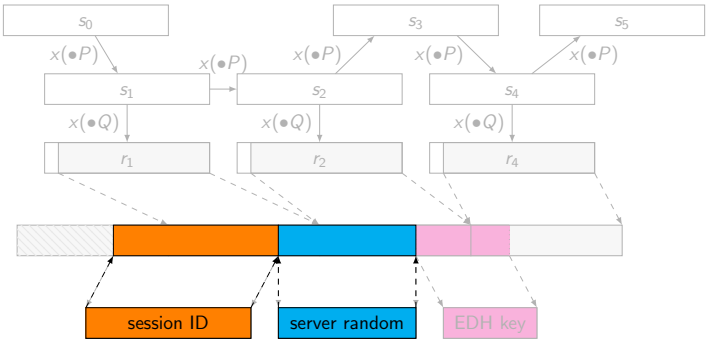
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



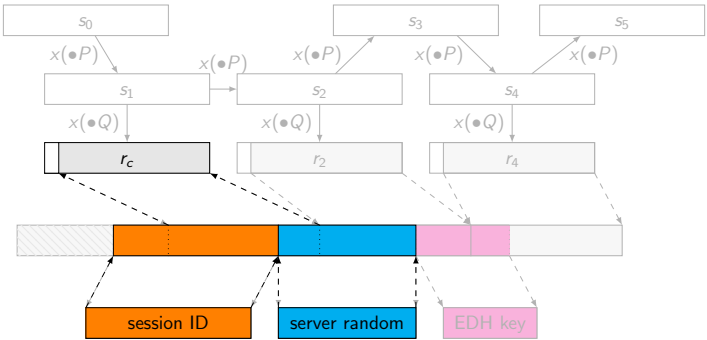
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



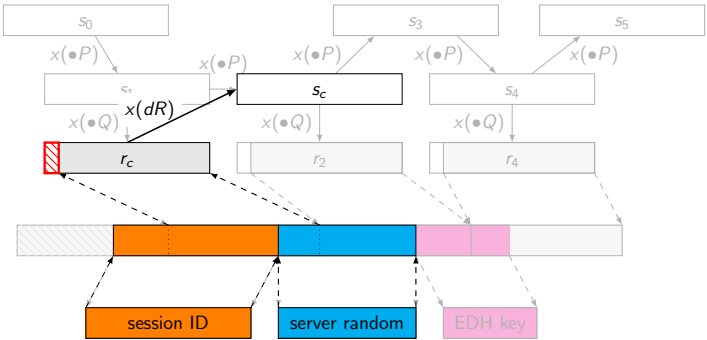
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



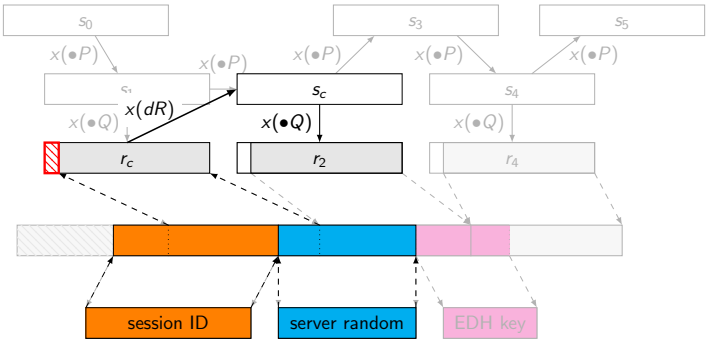
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



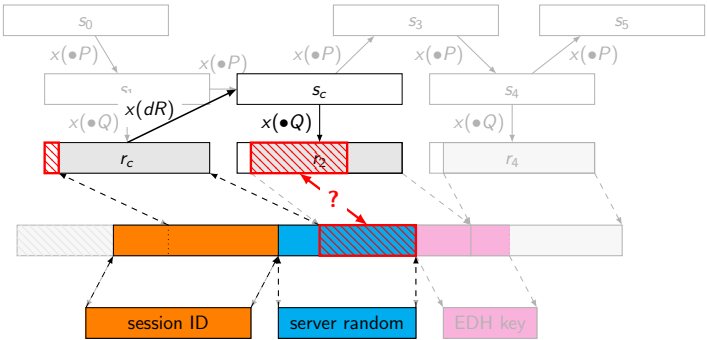
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



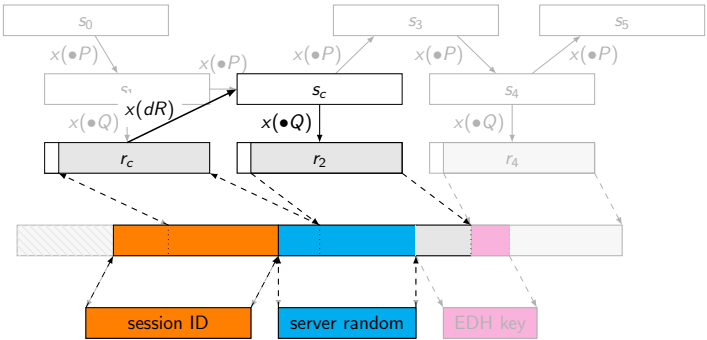
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



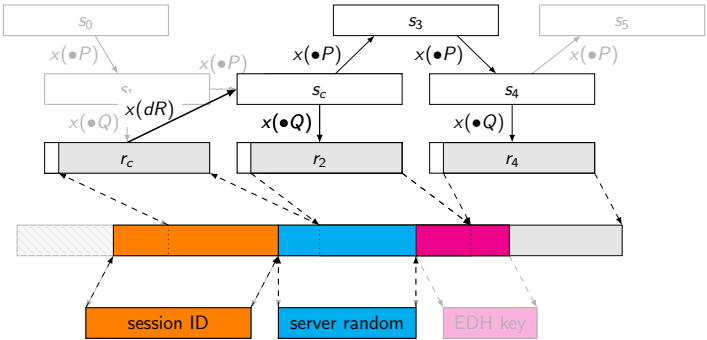
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



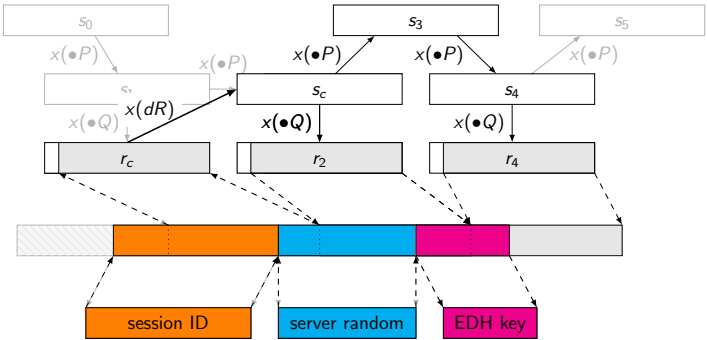
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



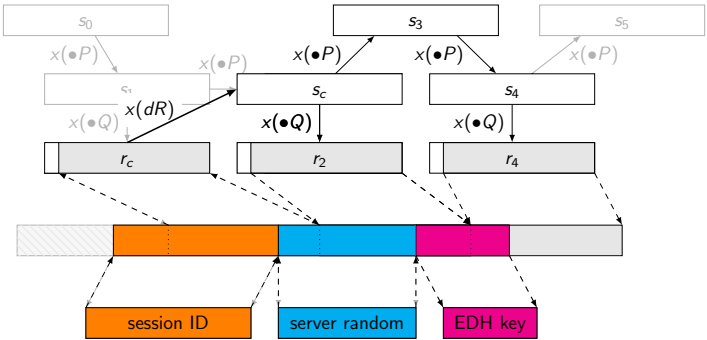
Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



Graphic thanks to Ruben Niederhagen.

Attack – Example: BSAFE-C



Graphic thanks to Ruben Niederhagen.

Timings

Attack	Intel Xeon Reference System		16-CPU AMD Cluster
	2^{22} Candidates (s)	Expected Runtime (min)	Total Runtime (min)
BSAFE-C v1.1	–	0.26	0.04*
BSAFE-Java v1.1	75.08*	641	63.96*
SChannel I	72.58*	619	62.97*
SChannel II	62.79*	1,760	182.64*
OpenSSL-fixed I	–	0.04	0.02*
OpenSSL-fixed II	–	707	83.32*
OpenSSL-fixed III	–	$2^k \cdot 707$	$2^k \cdot 83.32$

*measured

December 2013: Obama's NSA review panel report

Upon review, however, we are unaware of any vulnerability created by the US Government in generally available commercial software that puts users at risk of criminal hackers or foreign governments decrypting their data. Moreover, it appears that in the vast majority of generally used, commercially available encryption software, there is no vulnerability, or "backdoor," that makes it possible for the US Government or anyone else to achieve unauthorized access.[174]

[174] Any cryptographic algorithm can become exploitable if implemented incorrectly or used improperly.

And now to something completely different ...

Details on Intel's RNG

Details on Intel's RNG

[7] D. J. Johnston, "Mircoarchitecture Specification (MAS) for PP-DRNG," Intel Corporation (unpublished), V1.4, 2009.

[8] C. E. Dike, "3 Gbps Binary RNG Entropy Source," Intel Corporation (unpublished), 2011.

[9] C. E. Dike and S. Gueron, "Digital Symmetric Random Number Generator Mathematics," Intel Corporation (unpublished), 2009.

(References from "Analysis of Intel's Ivy Bridge Digital Random Number Generator Prepared for Intel" by Mike Hamburg, Paul Kocher, and Mark E. Marson. Cryptography Research, Inc.)

Design (from CRI report)

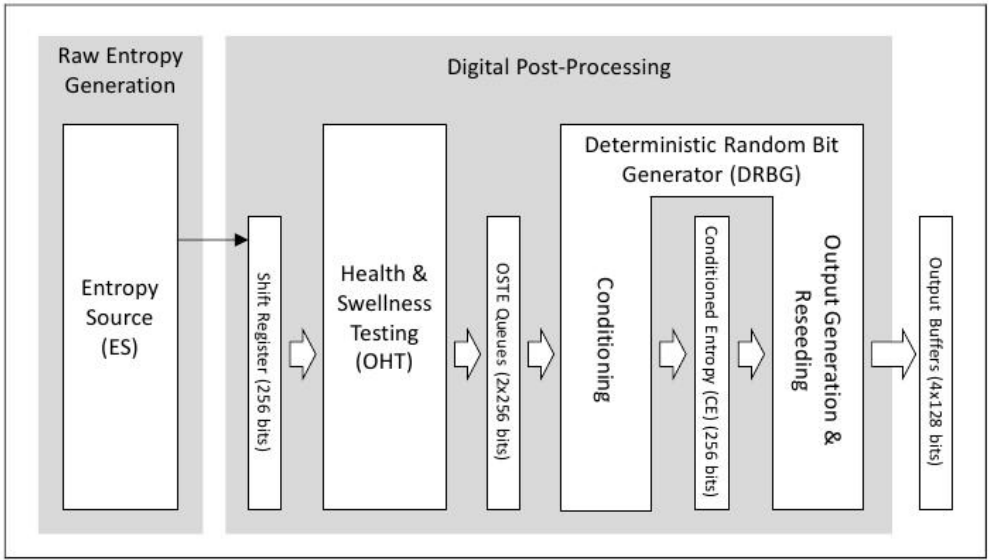
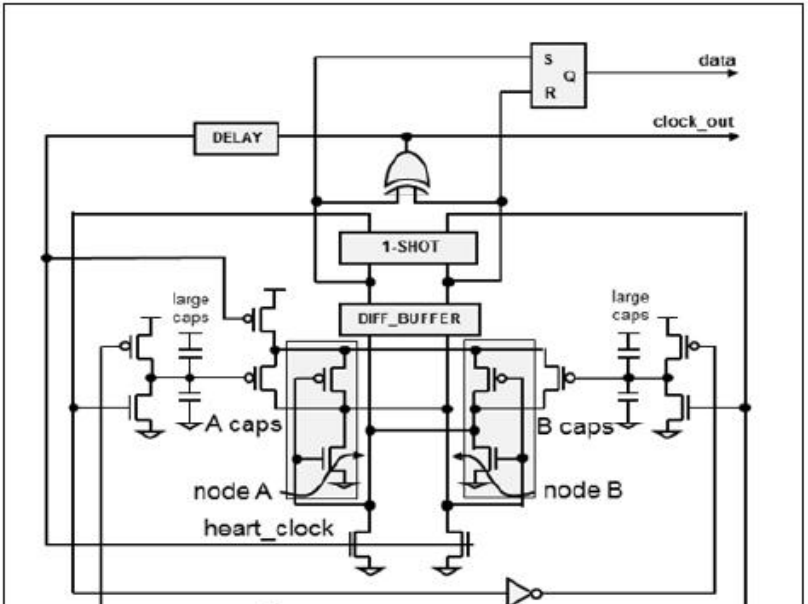
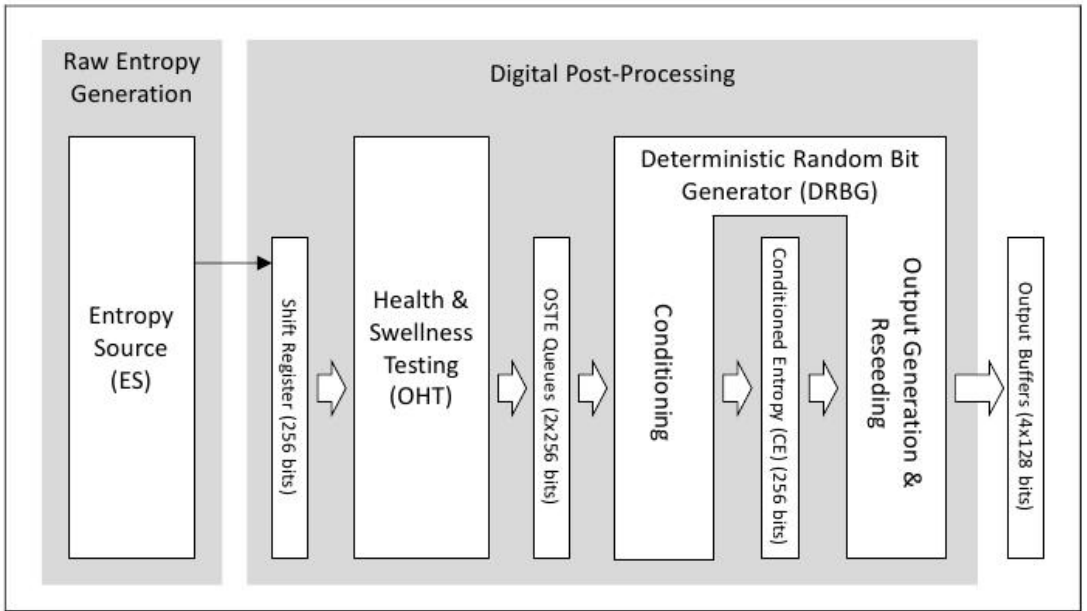


Figure 1: Block diagram of the Intel RNG (adapted from [7])

Entropy Source (from CRI report)



Design (from CRI report)



Intel assurances – David Johnston

I've examined my own RNG with electron microscopes and picoprobes. So I and a number of test engineers know full well that the design hasn't been subverted. For security critical systems, having multiple entropy sources is a good defense against a single source being subverted. But if an Intel processor were to be subverted, there are better things to attack, like the microcode or memory protection or caches. We put a lot of effort into keeping them secure, but as with any complex system it's impossible to know that you've avoided all possible errors, so maintaining the security of platforms is an ongoing battle. [..] But the implication at the top of this thread is that we were leaned on by the government to undermine our own security features. I know for a fact that I was not leant on by anyone to do that. X9.82 took my contributions and NIST is taking about half my contributions, but maybe they're slowly coming around to my way of thinking on online entropy testing. If I ultimately succeed in getting those specs to be sane, we better hope that I am sane.

Scary Paper of the Year: *Stealthy Dopant-Level Hardware Trojans*

by Becker, Regazzoni, Paar, and Burleson, CHES 2013

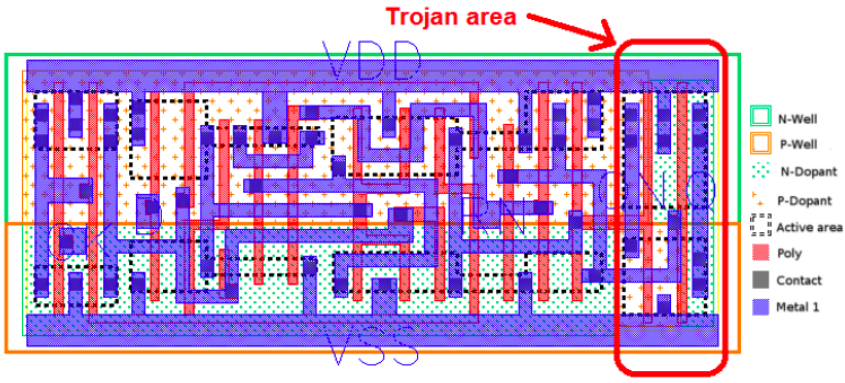


Fig. 2. Layout of the Trojan DFFR_X1 gate. The gate is only modified in the highlighted area by changing the dopant mask. The resulting Trojan gate has an output of $Q = V_{DD}$ and $QN = GND$.

Scary recommendations

CRI: “Because the Ivy Bridge RNG is implemented as an instruction in the CPU, it is much simpler to use than other hardware-based RNGs and avoids the need for additional software layers that could introduce bugs.”

Johnston: “Just use the output of the RDRAND instruction wherever you need a random number.” (github search for RDRAND has 33 609 code results)

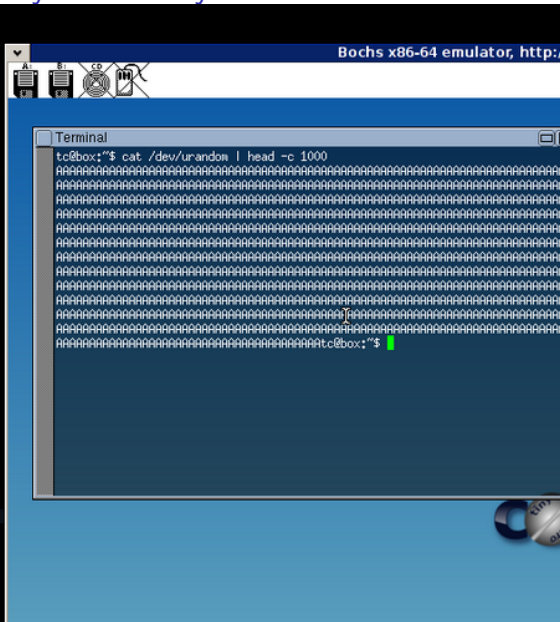
Intel manual 325462, June 2013, page 177:
"extremely rare cases" RDRAND "will return no data".
Also: "returning no data transitorily" because of "heavy load".
Recommendation to "retry for a limited number of iterations"; the subsequent explanation makes clear that this catches the "transitory" failures but not the "extremely rare" failures.
There is no quantification of "extremely rare".

Linux use of RDRAND

```
 -rw-r--r-- H. Peter Anvin     2012-07-27 22:26 random.c:
/*
* In case the hash function has some recognizable output
* pattern, we fold it in half. Thus, we always feed back
* twice as much data as we output.
*/
hash.w[0] ^= hash.w[3];
hash.w[1] ^= hash.w[4];
hash.w[2] ^= rol32(hash.w[2], 16);
/*
* If we have a architectural hardware random number
* generator, mix that in, too.
*/
for (i = 0; i < LONGS(EXTRACT_SIZE); i++) {
    unsigned long v;
    if (!arch_get_random_long(&v))
            break;
    hash.l[i] ^= v;
}
```

RDRAND backdoor proof of concept – Taylor Hornby

```
40   }
41 #endif
42
43   Bit16u val_16 = 0;
44
45   if (HW_RANDOM_GENERATOR_READY) {
46     val_16 |= rand() & 0xff;  // hack using std C rand() function
47     val_16 <<= 8;
48     val_16 |= rand() & 0xff;
49
50     setEFlagsOSZAPC(EFlagsCFMask);
51   }
52   else {
53     setEFlagsOSZAPC(0);
54   }
55
56   BX_WRITE_16BIT_REG(i->dst(), val_16);
57
58   BX_NEXT_INSTR(i);
59 }
60
61 BX_INSF_TYPE BX_CPP_AttrRegparmN(1) BX_CPU_C::RDRAND_Ed(bxInstruction_c *i)
62 {
63   Bit32u val_32 = 0;
64
65   BX_INFO(("In RDRAND_Ed!"));
66
67   Bit32u edx = get_reg32(BX_32BIT_REG_EDX);
68
69   if (EIP > 0xc0387d20 && EIP < 0xc0387e19) {
70     BX_INFO(("Triggering backdoor!"));
71     Bit32u at_edx = read_virtual_dword(BX_SEG_REG_DS, edx);
72     val_32 = at_edx ^ 0x41414141;
73   }
74
75   setEFlagsOSZAPC(EFlagsCFMask);
76
77   BX_WRITE_32BIT_REGZ(i->dst(), val_32);
78
79   BX_NEXT_INSTR(i);
80 }
81
82 #if BX_SUPPORT_X86_64
```



Updated in Linux repository (Dec 2013)

```
/*
* If we have an architectural hardware random number
* generator, use it for SHA's initial vector
*/
sha_init(hash.w);
for (i = 0; i < LONGS(20); i++) {
      unsigned long v;
      if (!arch_get_random_long(&v))
            break;
      hash.l[i] = v;
}
/* Generate a hash across the pool,
* 16 words (512 bits) at a time */
spin_lock_irqsave(&r->lock, flags);
for (i = 0; i < r->poolinfo->poolwords; i += 16)
      sha_transform(hash.w, (__u8 *)(r->pool + i), workspace);
```

Would you like to audit this? (State in early January)

```
2013-12-17 21:16 Theodore Ts'o      o [dev] [origin/dev] random: use the architectural HWRNG for~
2013-12-06 21:28 Greg Price         o random: clarify bits/bytes in wakeup thresholds
2013-12-07 09:49 Greg Price         o random: entropy_bytes is actually bits
2013-12-05 19:32 Greg Price         o random: simplify accounting code
2013-12-05 19:19 Greg Price         o random: tighten bound on random_read_wakeup_thresh
2013-11-29 20:09 Greg Price         o random: forget lock in lockless accounting
2013-11-29 15:56 Greg Price         o random: simplify accounting logic
2013-11-29 15:50 Greg Price         o random: fix comment on "account"
2013-11-29 15:02 Greg Price         o random: simplify loop in random_read
2013-11-29 14:59 Greg Price         o random: fix description of get_random_bytes
2013-11-29 14:58 Greg Price         o random: fix comment on proc_do_uuid
2013-11-29 14:58 Greg Price         o random: fix typos / spelling errors in comments
2013-11-16 10:19 Linus Torvalds     M-| Merge tag 'random_for_linus' of git://git.kernel.org/pub~
2013-11-03 18:24 Theodore Ts'o      | o [random_for_linus] random: add debugging code to detect ~
2013-11-03 16:40 Theodore Ts'o      | o random: initialize the last_time field in struct timer_r~
2013-11-03 07:56 Theodore Ts'o      | o random: don't zap entropy count in rand_initialize()
2013-11-03 06:54 Theodore Ts'o      | o random: printk notifications for urandom pool initializa~
2013-11-03 00:15 Theodore Ts'o      | o random: make add_timer_randomness() fill the nonblocking~
2013-10-03 12:02 Theodore Ts'o      | o random: convert DEBUG_ENT to tracepoints
2013-10-03 01:08 Theodore Ts'o      | o random: push extra entropy to the output pools
2013-10-02 21:10 Theodore Ts'o      | o random: drop trickle mode
2013-09-22 16:04 Theodore Ts'o      | o random: adjust the generator polynomials in the mixing f~
2013-09-22 15:24 Theodore Ts'o      | o random: speed up the fast_mix function by a factor of fo~
2013-09-22 15:14 Theodore Ts'o      | o random: cap the rate which the /dev/urandom pool gets re~
2013-09-21 19:42 Theodore Ts'o      | o random: optimize the entropy_store structure
2013-09-12 14:27 Theodore Ts'o      | o random: optimize spinlock use in add_device_randomness()
2013-09-12 14:10 Theodore Ts'o      | o random: fix the tracepoint for get_random_bytes(_arch)
2013-09-10 23:16 H. Peter Anvin     | o random: account for entropy loss due to overwrites
2013-09-10 23:16 H. Peter Anvin     | o random: allow fractional bits to be tracked
2013-09-10 23:16 H. Peter Anvin     | o random: statically compute poolbitshift, poolbytes, pool~
2013-09-21 18:06 Theodore Ts'o      | o random: mix in architectural randomness earlier in extra~
2013-11-11 12:20 Hannes Frederic S~ o | random32: add prandom_reseed_late() and call when nonblo~
2013-10-10 12:31 Linus Torvalds     M-| Merge tag 'random_for_linus' of git://git.kernel.org/pub~
2013-09-21 13:58 Theodore Ts'o      | o random: allow architectures to optionally define random_~
```

US 20070189527A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2007/0189527 A1**

Brown et al. (43) **Pub. Date: Aug. 16, 2007**

(54) **ELLIPTIC CURVE RANDOM NUMBER GENERATION**

(76) Inventors: **Daniel R. L. Brown**, Mississauga (CA); **Scott A. Vanstone**, Campbellville (CA)

Correspondence Address:
Blake, Cassels & Graydon LLP
Commerce Court West
P.O. Box 25
Toronto, ON M5L 1A9 (CA)

(21) Appl. No.: **11/336,814**

(22) Filed: **Jan. 23, 2006**

Related U.S. Application Data

(60) Provisional application No. 60/644,982, filed on Jan. 21, 2005.

Publication Classification

(51) **Int. Cl.**
H04L 9/00 (2006.01)

(52) **U.S. Cl.** **380/44**

(57) **ABSTRACT**

An elliptic curve random number generator avoids escrow keys by choosing a point Q on the elliptic curve as verifiably random. An arbitrary string is chosen and a hash of that string computed. The hash is then converted to a field element of the desired field, the field element regarded as the x-coordinate of a point Q on the elliptic curve and the x-coordinate is tested for validity on the desired elliptic curve. If valid, the x-coordinate is decompressed to the point Q, wherein the choice of which is the two points is also derived from the hash value. Intentional use of escrow keys can provide for back up functionality. The relationship between P and Q is used as an escrow key and stored by for a security domain. The administrator logs the output of the generator to reconstruct the random number with the escrow key.

Hat tip @nymble.

Snippets from the patent application

can provide for back up functionality. The relationship between P and Q is used as an escrow key and stored by for a security domain. The administrator logs the output of the generator to reconstruct the random number with the escrow key.

accounts. A more seamless method may be applied for cryptographic applications. For example, in the SSL and TLS protocols, which are used for securing web (HTTP) traffic, a client and server perform a handshake in which their first actions are to exchange random values sent in the clear.

[**0054**] Many other protocols exchange such random values, often called nonces. If the escrow administrator observes these nonces, and keeps a log of them **508**, then later it may be able to determine the necessary r value. This