On the Practical Exploitability of Dual EC in TLS Implementations

Stephen Checkoway[1], **Matt Fredrikson[2]**, Ruben Niederhagen[3], Matt Green[1], Tanja Lange[3], **Tom Ristenpart[2]**, Dan Bernstein[3,4], Jake Maskiewicz[5], Hovav Schacham[5]

Johns Hopkins[1], University of Wisconsin[2], TU Eindhoven[3], University of Illinois — Chicago[4], UCSD[5]

Background

History

- Dual EC is a *deterministic random bit generator* included in NIST SP 800-90 until April 2014
- Leaked documents led many to believe that Dual EC contains a *backdoor* known to intelligence agencies

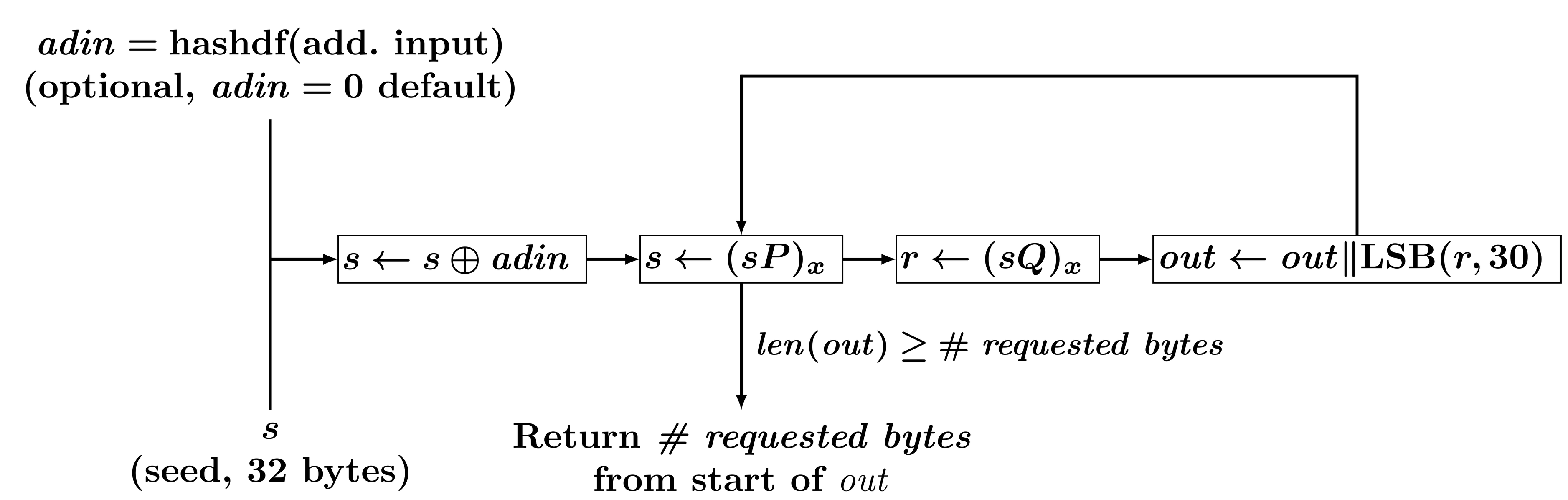
Backdoor

- Suggested by Shumow and Ferguson, 2007
- Dual EC based on points *P* and *Q* (*P* is prime-order generator)
- …so there exists a constant *d* such that $dQ = P$
- Outputs correspond to the *x*-coordinate of internal state *s*, multiplied by *Q* (i.e., $out = \mathbf{LSB}[(sQ)_x, 30]$)
- Knowledge of *d* sufficient to learn the *next* state *s* from *out*

Attack

1. Guess 2 most-significant bytes of output to get $(sQ)_x$
2. Multiply by *d*, i.e., $d(sQ) = sP$
3. $(sP)_x$ is the next internal state

Complexity is ~2^{15}

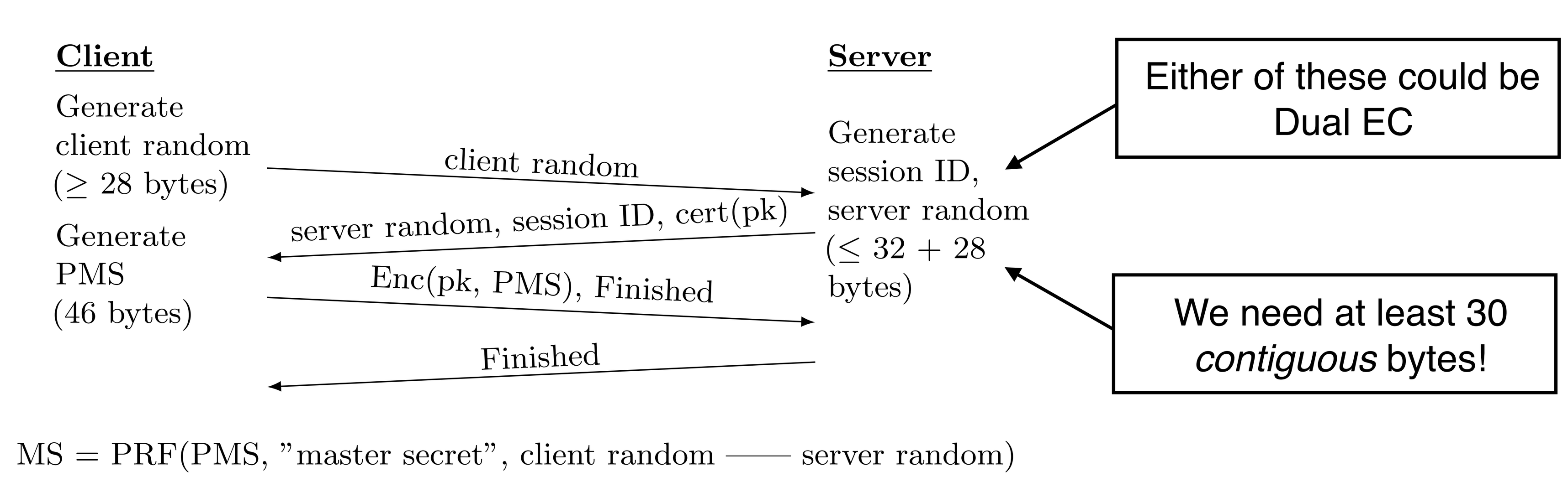


Is it exploitable?

In practice, an actual implementation might…

- Not release enough random data in the clear
- Mix additional sources of random data into state/output
- Use unpredictable interleavings of calls to Dual EC
- Cache unused partial output blocks
- Aggressively re-seed the internal state
- Implement the spec incorrectly (this happened — *twice!)*

In short, implementation details matter — the backdoor is fragile



Methodology

Goal: understand whether variants of Shumow-Ferguson attack work on real TLS implementations

- Studied three commercial/open source implementations: RSA's BSAFE, Microsoft's SChannel, and OpenSSL-FIPS
- Assume a passive network adversary who knows the backdoor constant *d* such that $dQ = P$
- "Implemented" this assumption by modifying implementations to use a new value for *Q*, for which we know *d*
  - Modified OpenSSL-FIPS source to encode new *Q*
  - Reverse-engineered SChannel, BSAFE-Java, BSAFE-C to overwrite *Q*, disable known-answer tests
- Instantiated servers using OpenSSL-FIPS (Apache), BSAFE, and SChannel (IIS), as well as a client using SChannel (IE)
- Captured packet traces using Wireshark, attempted to derive session keys

Results

Library	Default PRNG	Cache Output	Ext. Random	Bytes per Session	Adin Entropy	Attack Complexity	Time (minutes)
BSAFE-C v1.1	✓	✓	✓†	31–60	—	$30 \cdot 2^{15}(C_v + C_f)$	0.04
BSAFE-Java v1.1	✓		✓†	28	—	$2^{31}(C_v + 5C_f)$	63.96
SChannel I‡				28	—	$2^{31}(C_v + 4C_f)$	62.97
SChannel II‡				30	—	$2^{33}(C_v + C_f) + 2^{17}(5C_f)$	182.64
OpenSSL-fixed I*				32	20	$2^{15}(C_v + 3C_f) + 2^{20}(2C_f)$	0.02
OpenSSL-fixed III**				32	$35 + k$	$2^{15}(C_v + 3C_f) + 2^{35+k}(2C_f)$	$2^k \cdot 83.32$

* Assuming process ID and counter known. ** Assuming 15 bits of entropy in process ID, maximum counter of 2^k.
† With a library–compile-time flag. ‡ Versions tested: Windows 7 64-bit Service Pack 1 and Windows Server 2010 R2.

- Experiments performed on a four-node, quad-socket Opteron 6276 cluster
- C_v is a variable-base scalar multiplication, C_f is a fixed-base multiplication
- Times refer to attack on a *single* session
- For all but BSAFE-C, *dragnet surveillance is unlikely*
- *Targeted* surveillance is possible for all tested implementations

We performed a ZMap scan of 38 million servers

- Only 720 were running BSAFE
- 2.7 million were running SChannel

BSAFE Instances by Country