Stone Knives and Bear Skins: Why does the Internet still run on pre-historic cryptography?

Eric Rescorla
ekr@rtfm.com

Indocrypt 2011

Overview

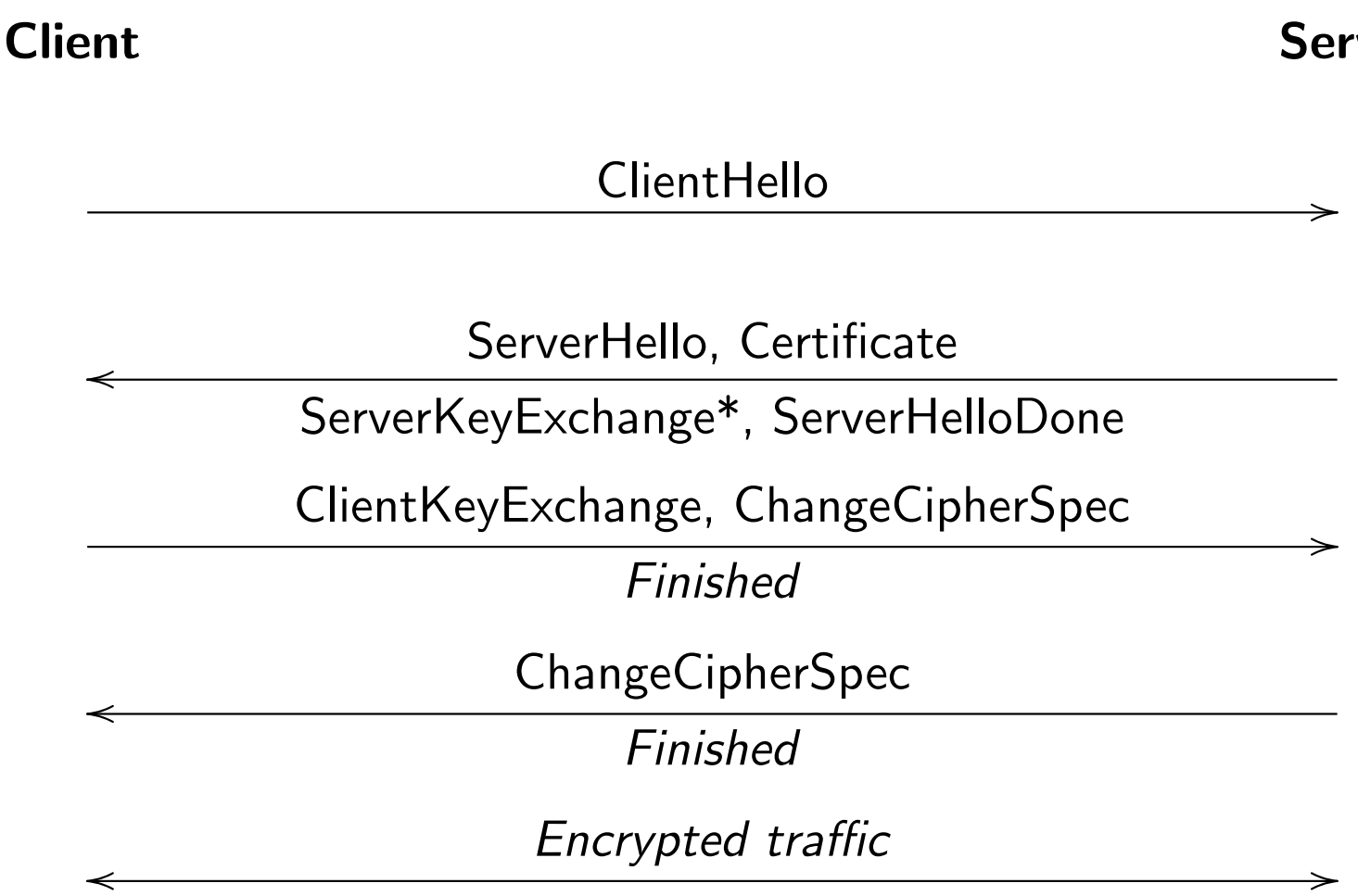
- Our cryptographic protocols use ancient algorithms
- In many cases these algorithms have known flaws
  - Or are used in ways known to be unsafe
- Why don't people upgrade?
  - Easier to patch
  - Backward compatibility
  - Collective action problems
  - Confusing security implications
- This turns out to be surprisingly hard to fix

Important Internet Security Protocols (Personal View)

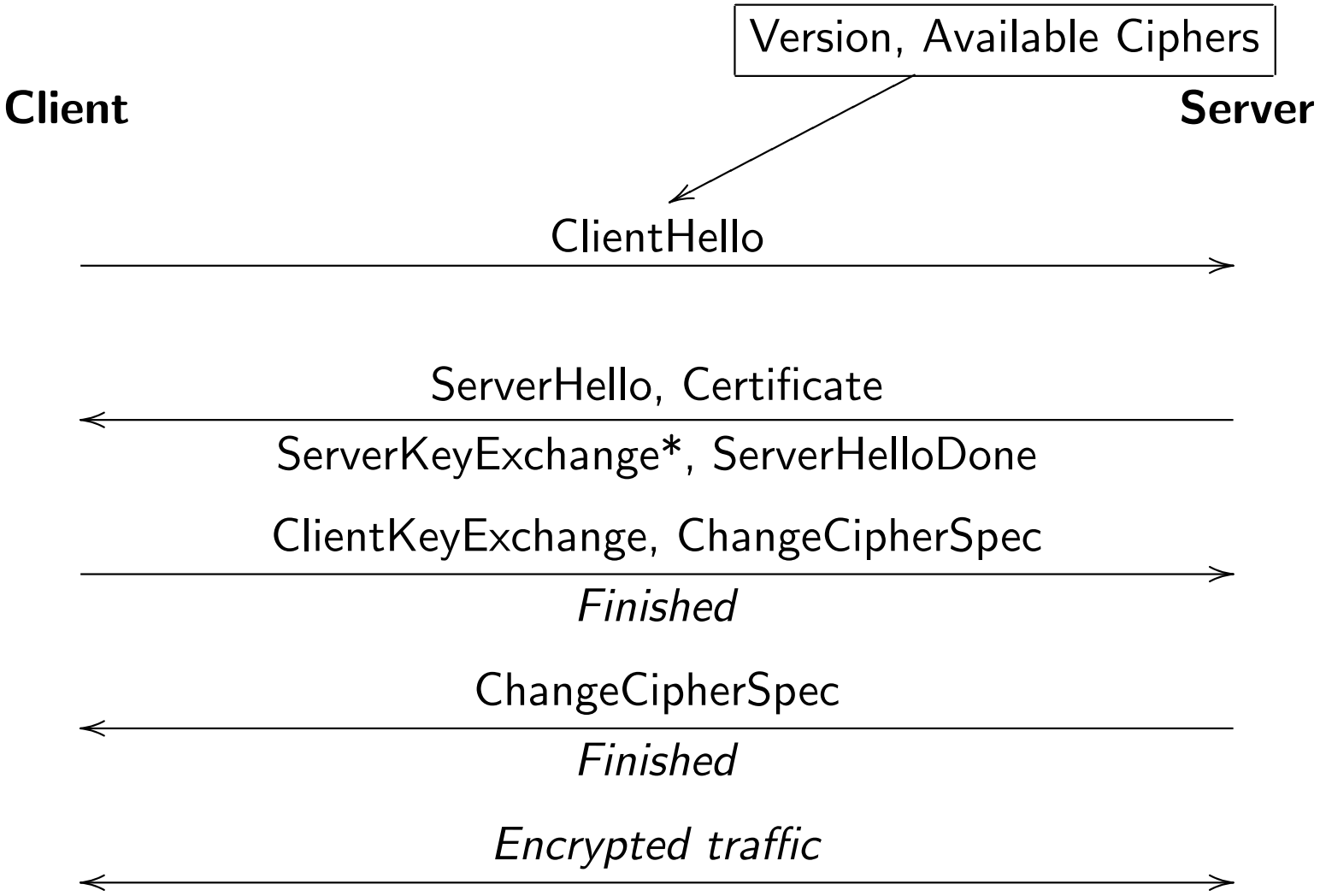
- SSL/TLS (really HTTPS)
- OAuth
- IPsec
- SSH
- S/MIME, PGP, ...

- I will mostly be talking about SSL/TLS
  - But these comments apply more generally

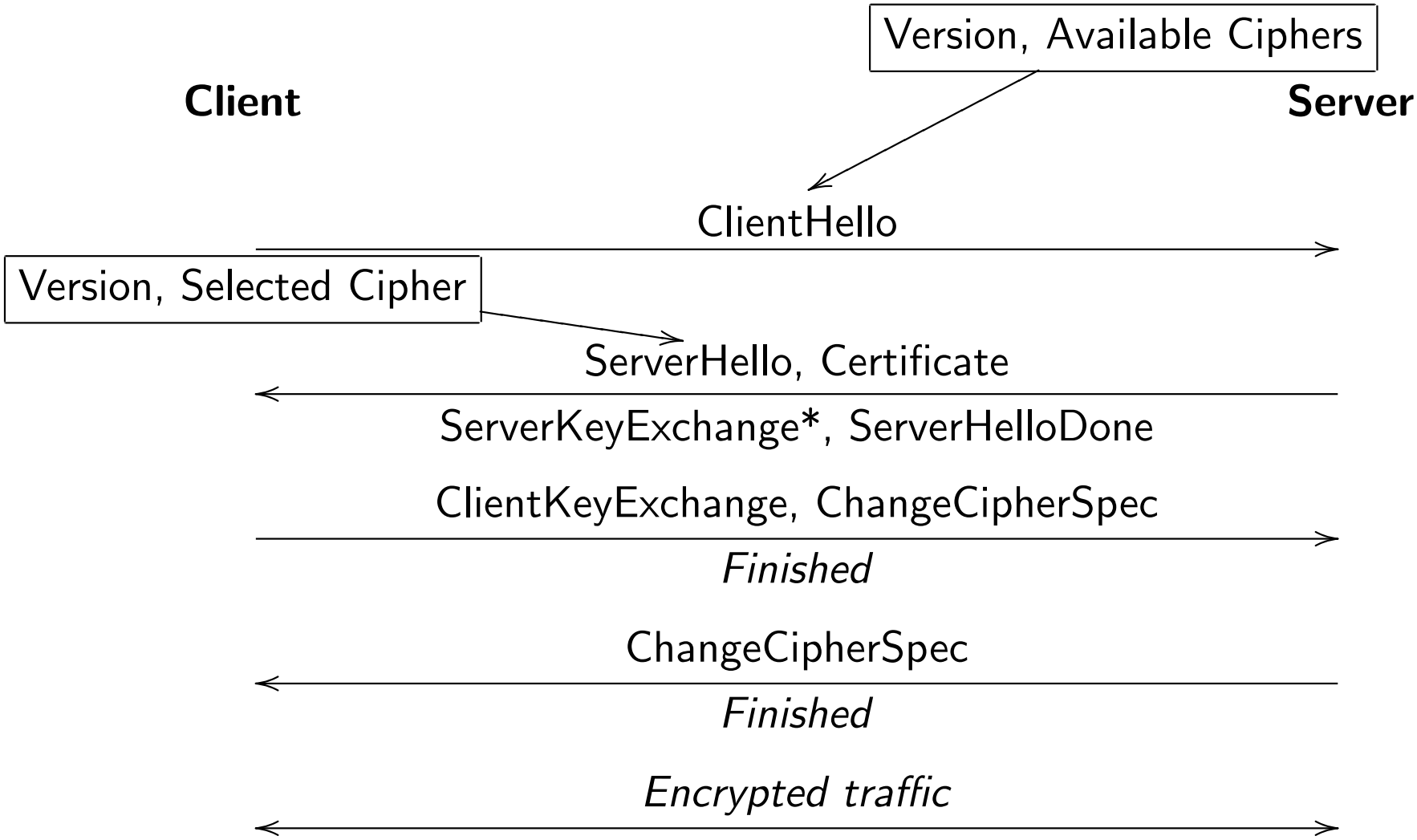
Review: The SSL/TLS Handshake [DR08]



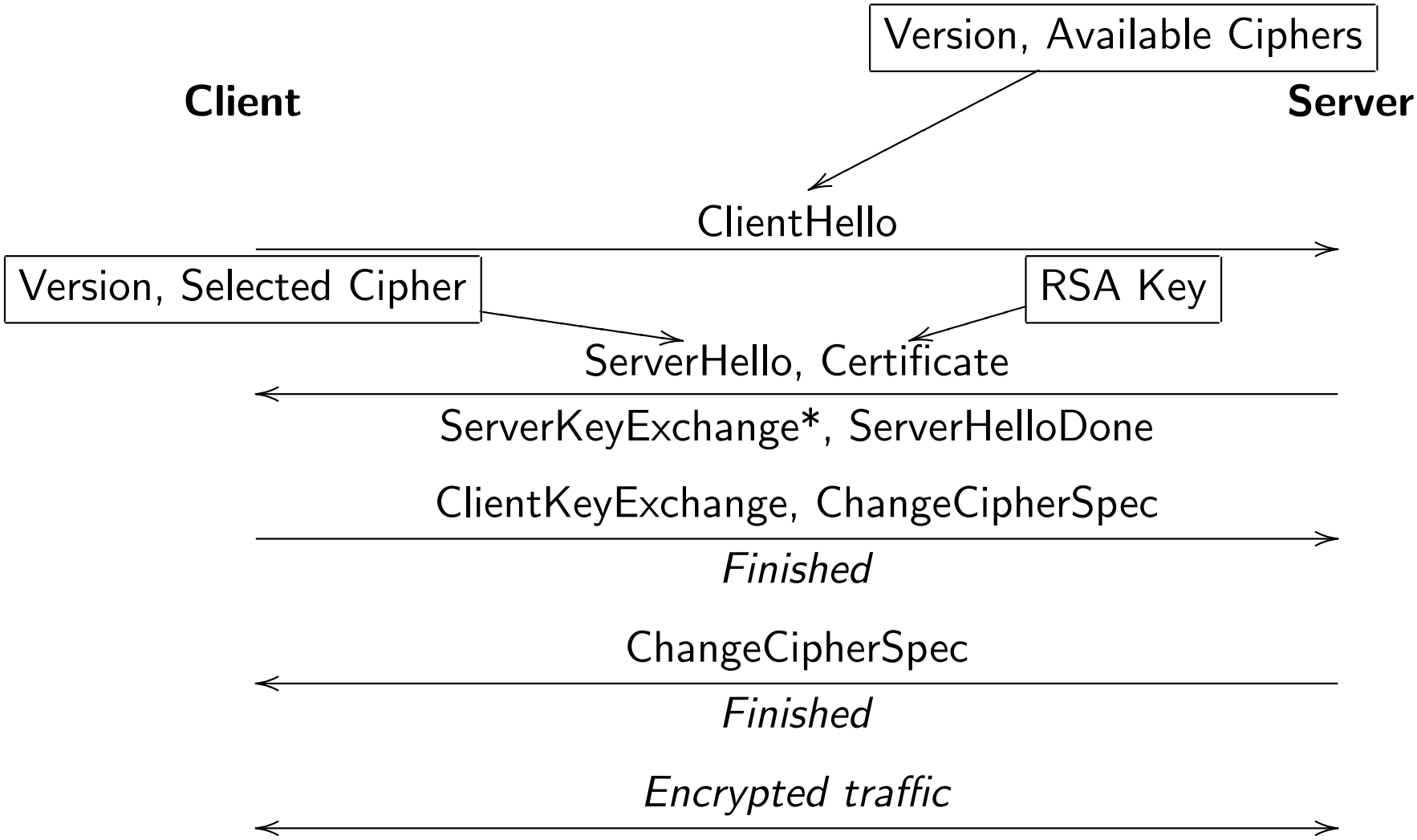
Review: The SSL/TLS Handshake



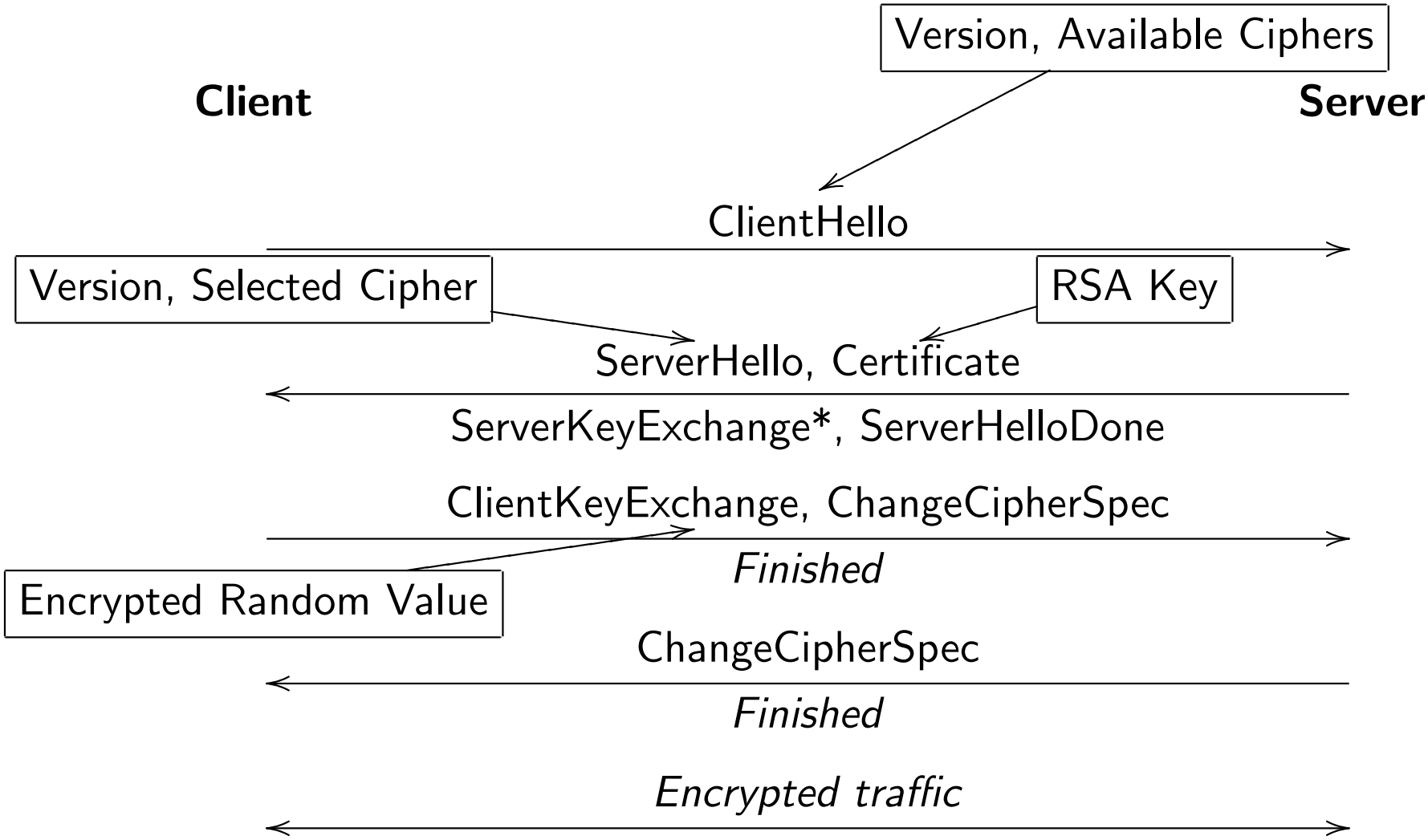
Review: The SSL/TLS Handshake



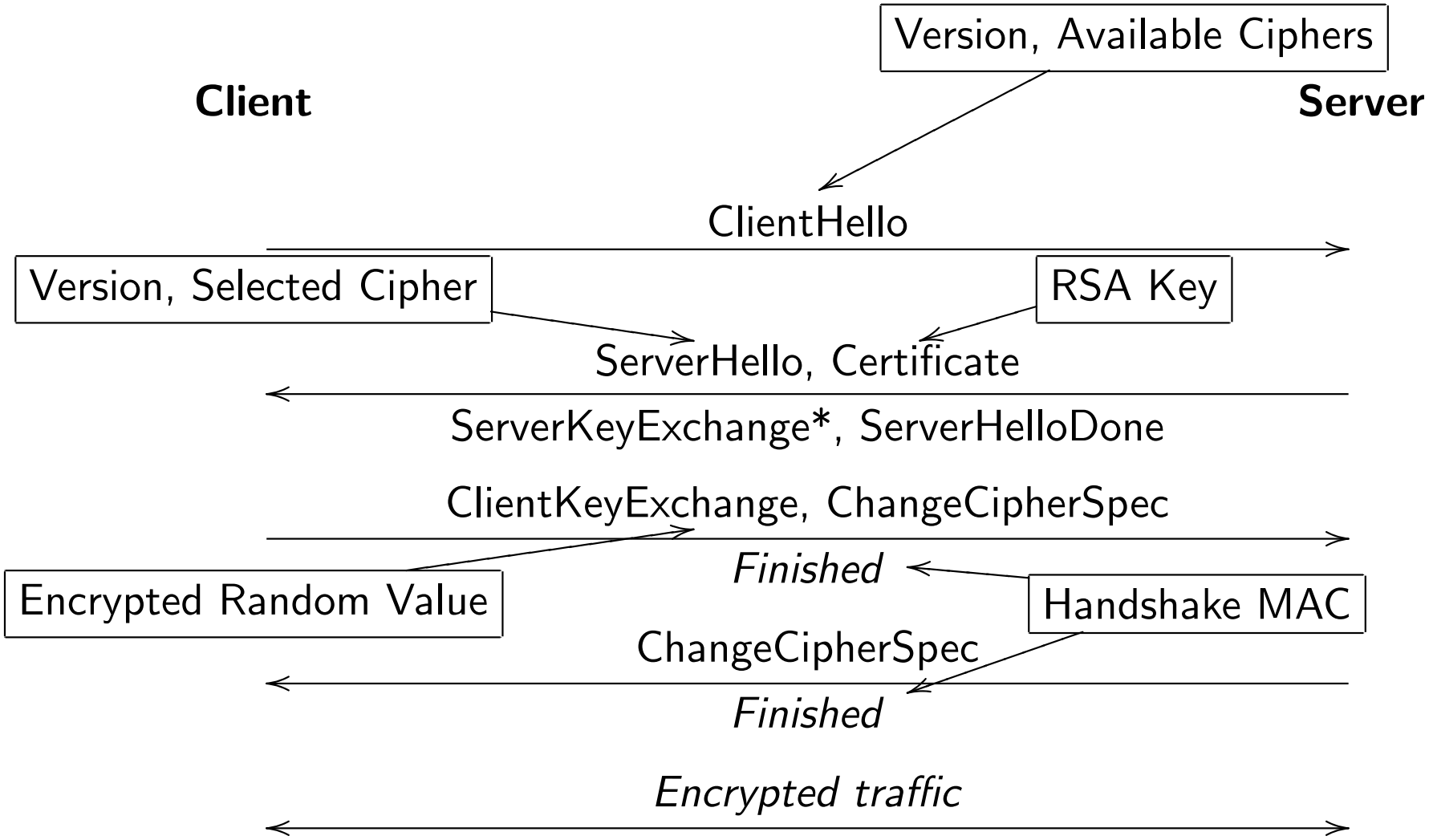
Review: The SSL/TLS Handshake



Review: The SSL/TLS Handshake



Review: The SSL/TLS Handshake

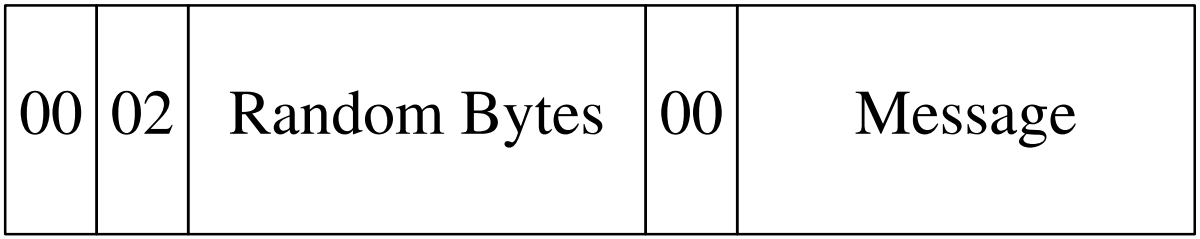


RSA Key Exchange

- TLS uses PKCS#1 v1.5 [JK03]
  - Over twenty years old
- Arguably superior alternatives exist
  - Optimal Asymmetric Encryption Padding [BR96]
  - Actually defined in RFC 3447

  "Two encryption schemes are specified in this document: RSAES-OAEP and RSAES-PKCS1-v1_5. RSAES-OAEP is recommended for new applications; RSAES-PKCS1-v1_5 is included only for compatibility with existing applications, and is not recommended for new applications." [JK03]

Reminder: PKCS#1 Padding



PKCS#1 1.5 + TLS was actually insecure

- Bleichenbacher [Ble98] and Klima show adaptive chosen ciphertext attacks
- Basic idea: use server as a PKCS#1 formatting oracle
  - Capture one ClientKeyExchange (RSA-encrypted secret)
  - Send modified versions and see if the server believes they are correctly formatted
    - Server generates an "alert" message if not
  - Triangulate in on original plaintext (takes about 2^{20} messages)
- OAEP resists this attack
  - Chance of random generating a correctly formatted message is very low

TLS Countermeasures to Bleichenbacher Attack

- Don't generate an error with incorrectly formatted data
  - Instead, randomly generate a fake random master secret and proceed with the handshake
  - What would have happened if the CKE was correctly formatted
- Handshake will fail at the Finished stage
  - Client and server don't share the same keys

Why this solution?

- Initially, this countermeasure was faster
  - Could be implemented solely on the server side
  - Didn't require any change to the client
  - Server side countermeasure needed in any case
    - Some clients won't upgrade
- Why didn't later versions of TLS adopt OAEP?
  - Inertia
  - Lack of clear benefit
  - Need for protocol mechanisms to negotiate it

Digression: Negotiating Protocol Versions

- All agents start out supporting version n
- We want to introduce $n+1$ while retaining backward compatibility
  - Need some kind of version negotiation mechanism
- Problems
  - How to negotiate securely?
  - When can you discard version n? (probably never)

Client Version	Server Version	
	n	$n/n+1$
n	n	n
$n/n+1$	n	$n+1$

Table 1: Desired Negotiation Outcome

SSL/TLS Negotiation Mechanisms

- Version number
- Cipher suites
- "certificate types" field in CertificateRequest
- Extensions (published in 2003) [BWNH+03]

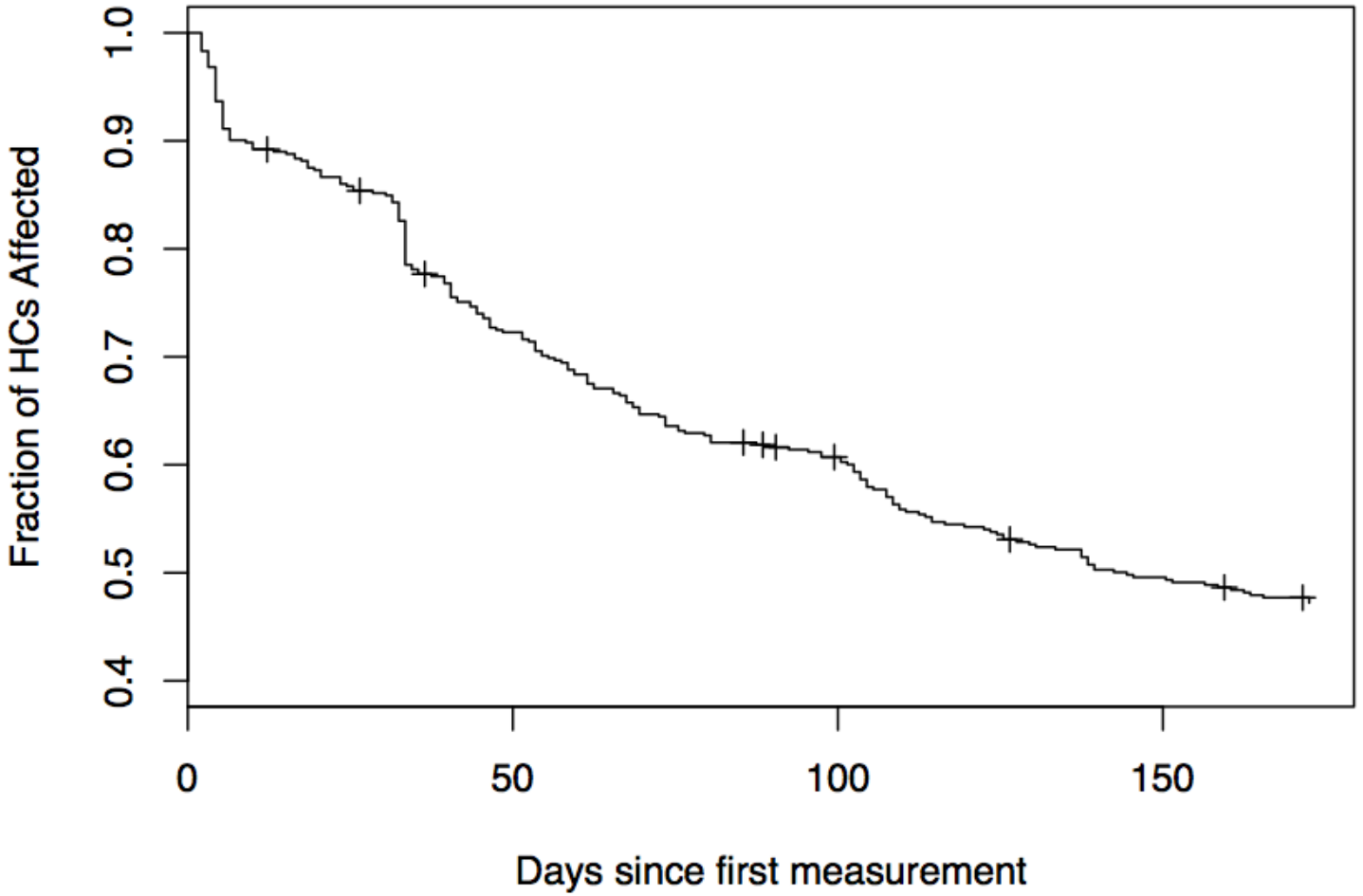
Options for Introducing OAEP

- Have clients unilaterally use OAEP
  - This creates obvious breakage
- Do an entirely new version of SSL/TLS
  - Very heavyweight
  - Not politically feasible as TLS 1.0 was still in process
- Introduce new RSA cipher suites that mean OAEP
  - Only really workable solution
  - Pretty ugly
- Remember: protection still needed for old cipher suites

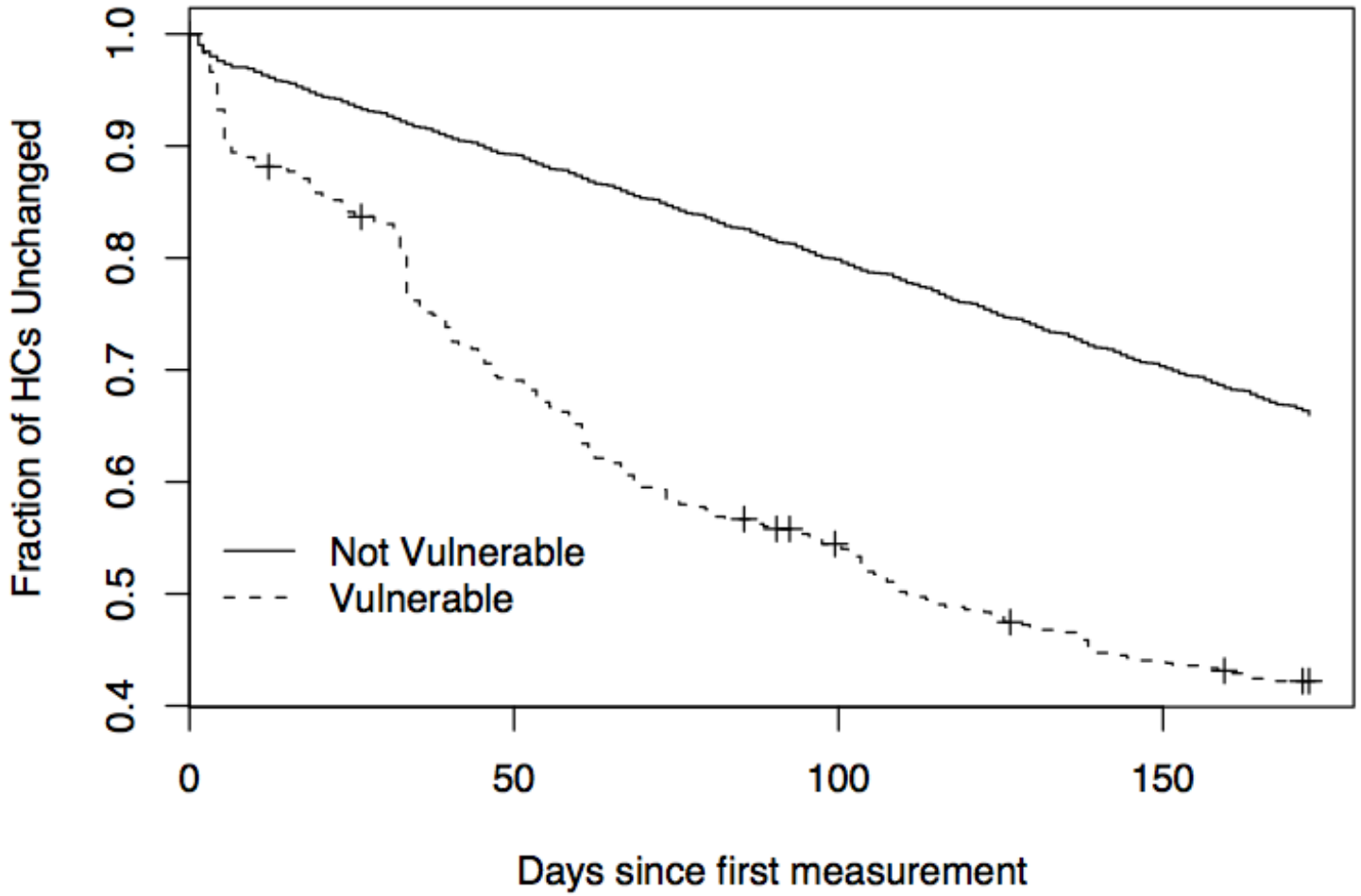
The Best (Worst?)-Case Scenario: Debian PRNG

- In 2006, Debian version of OpenSSL patched to fix Valgrind warnings
  - Accidentally wipes out nearly all entropy in PRNG ($\tilde{1}6$ bits left)
- Noticed in 2008 by Luciano Bello
- About 1% of servers had predictable private keys
  - Easily remotely detectable
  - Completely breaks RSA cipher suites against passive attack
  - Breaks DHE cipher suites against active attack or fancy passive attack [YRS+09]
- Imperative that servers fix
  - Fix was compatible and easy (get a new certificate)

How fast did people fix affected servers? [YRS+09]



Certificate churn versus natural replacement rate



Certificate Signature Algorithms

- Nearly all certificates are signed with either
  - RSA PKCS#1 1.5 with MD5 [Riv92]
  - RSA PKCS#1 1.5 with SHA-1 [NIS02] (Designed in 1993)
- Virtually no use of PSS
- Minimal use of SHA-256 or greater

Security of MD5 and SHA-1

- MD5 is completely broken against collisions
  - Work factor is about 2^{24}
  - Preimage attacks still impractical
- SHA-1 is severely compromised
  - Status is a little unclear (at least to me)
  - Collision factor appears to be about 2^{63} (better results seem to be unconfirmed)
  - No actual collisions have been found
- Even collisions are a threat to certificates [SSA+09]
  - Though countermeasures (sequence number randomization) are available

New hashes, old protocols

- How do we transition to a new algorithm?
- Authenticating parties need *two* certificates
  - One for SHA-1 (or maybe MD5)
  - One for SHA-x
- Relying parties need to support SHA-1 *and* SHA-x
  - Until nearly all authenticating parties have SHA-x certificates

A certificate with a strong hash doesn't help ME

- Threat is an attacker getting a certificate in my name
  - The existence of a weak certificate for me doesn't help them
  - Modifying that certificate requires a second preimage attack
    - Existing attacks involve (easier) collision-finding
    - Easier to modify certificate name rather than public key?
- I want *relying parties* to stop accepting weak hashes
  - But my actions don't really affect that
- Classic collective action problem

Hash algorithm transitions in SSL/TLS [BR06]

- Problem: incremental deployment of new hash functions
  - "Easy" part: certificates
  - Hard part: MD5 and SHA-1 are hardwired into TLS before 1.2
- This turns out to be a huge hassle
  - Design finished in 2008
  - Only starting to roll out now

Negotiating Certificate Digests

- This should be easy
  - Certificates have a hash algorithm field
  - TLS has negotiable cipher suites
    - They have a digest in them
    - E.g., `TLS_RSA_WITH_RC4_128_SHA`
- TLS cipher suites don't control the certificate digest
  - No way for clients to indicate that they support SHA-256
  - So only safe to send MD5 and SHA-1 certs
- Solution: signature_algorithms extension
  - Indicates which signature and digest algorithms each side supports

Replacing the TLS PRF

- TLS before 1.2 had a hardwired internal PRF
  - Used for key generation and handshake integrity check
  - Based on MD5 and SHA-1 XORed together
- This is probably safe
  - But still pretty scary
- TLS 1.2 has a negotiable PRF
  - Tied to the cipher suite
  - Default is SHA-256
- Note: security of the handshake is now no stronger than HMAC-SHA256

Multiple hash algorithms in S/MIME

- Store and forward systems are even harder
  - You have no way of knowing the peer's capabilities
- Only real solution is to use both algorithms in parallel
  - E.g., sign with SHA-1 *and* SHA-256
- Now there are concerns about signature stripping
  - S/MIME has an attribute designed to defend against this
  - But only works with collision-based attacks

Something that went fairly well: transition to AES

- SSL/TLS, S/MIME, IPsec etc. were designed before AES
  - Typical ciphers supported: DES, 3DES, RC4, RC2
- AES protocol support added rapidly
  - AES [NIS01] published in 2001
  - AES for TLS published in 2002 [Cho02]
  - AES for CMS published in 2003 [Sch03]
  - AES for IPsec published in 2003 [FGK03]
- Implementations kept pace with standardization
  - OpenSSL added AES for TLS and S/MIME in 2002

SSL/TLS AES deployment far from universal

- Many popular sites still prefer RC4 to AES
  - Examples: Google, Wells Fargo, Amazon
  - AES still only supported in $< 65\%$ of servers[Ris11]
- RC4 is 1.5-2x faster than AES-CBC
- No *practical* known security problems with RC4
  - Though lots of concerns about apparent non-randomness
  - Especially with the initial bytes

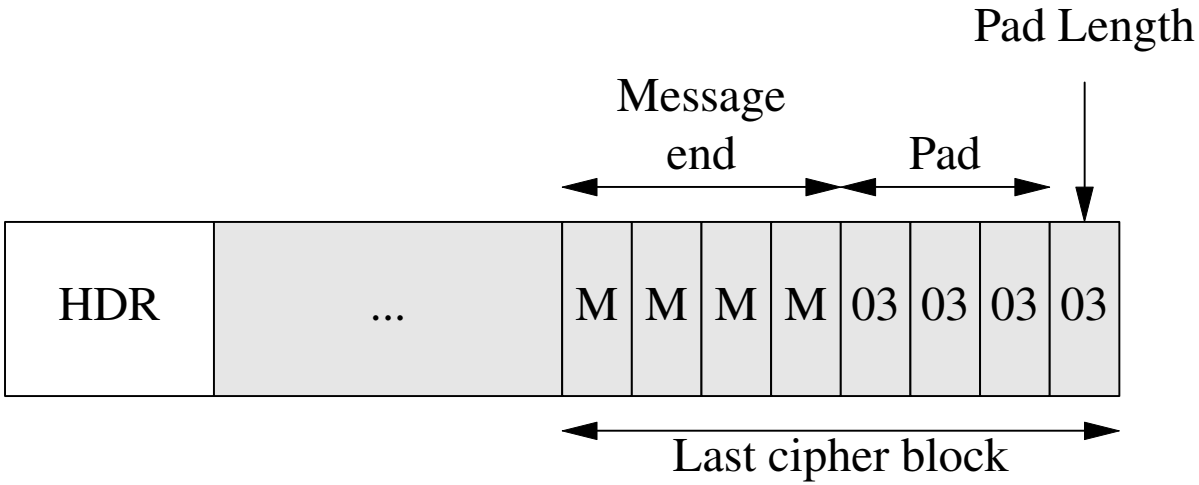
Why did AES deployment go as well as it did?

- Protocols were designed for encryption algorithm agility
  - The one thing everyone knew they needed to be able to replace
  - AES has nearly exactly the same "API" as DES
    - Except for block and key size
  - In many cases implementations automatically negotiated AES support
- AES filled a real gap
  - 3DES obviously too slow
  - Concerns about security of RC4
- Very strong push by USG
  - Many applications where AES is required

Cipher Block Chaining Issues

- SSL/TLS, IPsec, and S/MIME all designed around CBC
- SSL/TLS is especially bad here
  - Implicit (inter-packet) IV until TLS 1.1
  - Authenticate then Encrypt (AtE), not Encrypt then Authenticate (EtA) [Kra01]
- Several known attacks on CBC as used in SSL/TLS
  - Attacks on the padding [CHVV03]
    - Fixed with countermeasures
  - Attacks based on predictable IVs [Moe]
    - Clumsy countermeasures
    - Repaired in TLS 1.1 [DR06] and DTLS [MR04]

CBC Padding Reminder (SSL/TLS variant)



CBC Attack [CHVV03]

- Capture cipher block pair C_i, C_{i+1}
  - Our target is block C_{i+1}
  - Specifically the last byte of block C_{i+1}
- To test our guess for the last byte $== X$, we generate a record ending in
  - $C_i \oplus X, C_{i+1}$
- If we are right, we end up with the last byte $==$ 0 (no padding)
  - The MAC check now fails
  - But SSL/TLS has a different error message for padding and MAC errors
  - And there is a timing analysis variant

Defenses to CBC attack

- "Right" approach is to go to EtA
  - This blocks record tampering attacks
- Instead SSL/TLS uses a bunch of countermeasures
  - Return *mac_error* for all decoding errors
  - Bogus MAC check to hide timing channel
- Why?
  - Attack is very inefficient
    - SSL/TLS terminates after decoding errors (but DTLS...)
  - Unilateral versus bilateral changes

Review of Predictable IV attacks

- Scenario: Attacker can observe ciphertext and inject his own plaintext
  - He observes a block C_i and wants to verify his guess X for its value
- Attacker sees a record with trailing block B
  - This means that B is the IV for the next block
- Attacker injects $C_{i-1} \oplus B \oplus X$ as plaintext
  - Victim encrypts $B \oplus C_{i-1} \oplus B \oplus X = C_{i-1} \oplus X$
  - If result is C_i then the guess was correct

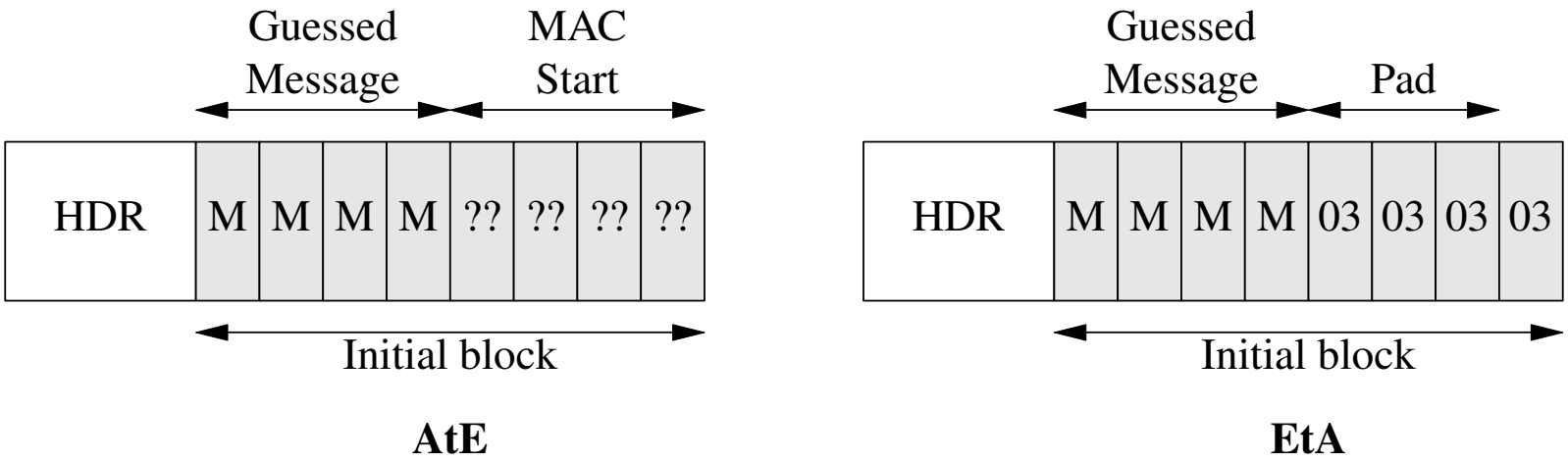
Limitations of Predictable IV Attacks

- Need tight control of the channel
  - Didn't seem likely except for VPN settings
- Need to guess an entire block at a time
  - Not easy!
- This all doesn't sound very serious
  - TLS WG duly fixed TLS [DR06]
  - But practically nobody implemented it

Predictions are hard... especially about the future

- Rizzo/Duong "BEAST" paper changed people's perceptions of the risk
  - New technique for byte-by-byte guessing
  - New threat vector via Web technologies (WebSockets and Java)
- But this was fixed in TLS 1.1
  - So we'll just deploy TLS 1.1, right?
  - Well sort of...
- People are deploying TLS 1.1
  - But *also* 1/n+1 splitting countermeasure
  - And active attacks on TLS version negotiation are possible

Irony Alert: AtE makes IV-prediction attacks harder



- Easier to guess short messages
- But the MAC is appended to the message before encryption
  - And is hard to guess
- With EtA the MAC isn't encrypted
  - And the padding is predictable

Implementing for extensibility is really hard

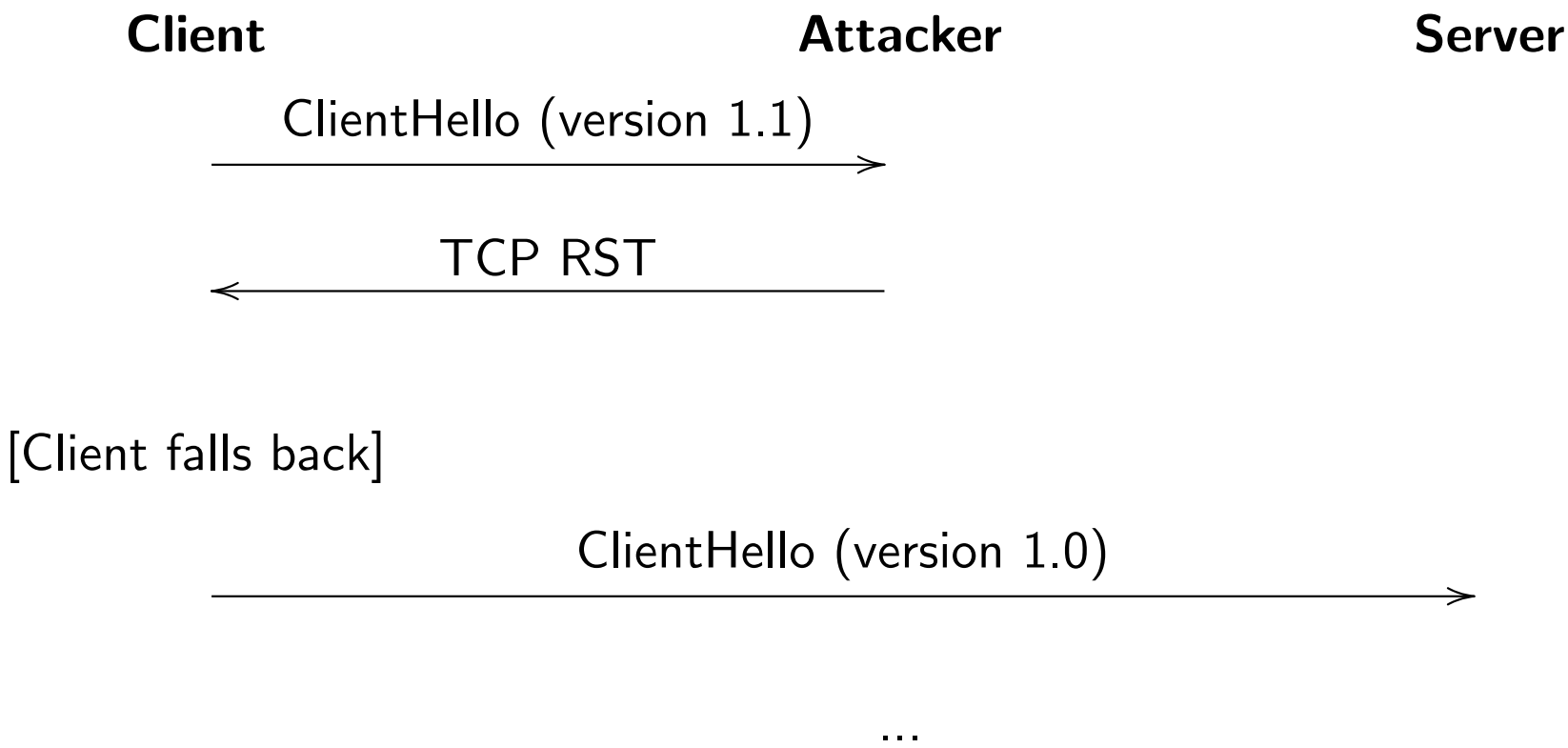
"Note: some server implementations are known to implement version negotiation incorrectly. For example, there are buggy TLS 1.0 servers that simply close the connection when the client offers a version newer than TLS 1.0. Also, it is known that some servers will refuse the connection if any TLS extensions are included in ClientHello. Interoperability with such buggy servers is a complex topic beyond the scope of this document, and may require *multiple connection attempts by the client.*[*]

Earlier versions of the TLS specification were not fully clear on what the record layer version number (TLSPlaintext.version) should contain when sending ClientHello (i.e., before it is known which version of the protocol will be employed). Thus, TLS servers compliant with this specification MUST accept any value 03,XX as the record layer version number for ClientHello.

TLS clients that wish to negotiate with older servers MAY send any value 03,XX as the record layer version number. Typical values would be 03,00, the lowest version number supported by the client, and the value of ClientHello.client_version. No single value will guarantee interoperability with all old servers, but this is a complex topic beyond the scope of this document." [DR08]

[*]emphasis mine

A simple downgrade attack



- Attacker pretends to be a broken server
- This bypasses all TLS anti-downgrade mechanisms

How should I be encrypting data anyway?

- One motivation for CBC was that stream ciphers seemed hard to get right
  - Concerns about keystream/IV reuse
  - Ridiculously weak integrity properties
- But now it seems CBC is hard to get right too
  - Oops...
- Some emerging consensus on AEAD algorithms
  - GCM is the model here
  - ... but finding new risks with stream ciphers [WMSM11]
- Lack of clean layer separation between COMSEC and crypto

Why is it so hard to enhance fielded systems?

- Many of the extension points aren't
  - Code (or standards) which hasn't been tested doesn't work
  - ... any new primitive needs to look exactly like an existing primitive
- Changes in only one side are easier
  - But this generally precludes protocol/algorithm changes
  - And needed anyway to support older peers
- Hard to evaluate the security impact of cryptographic issues
  - Cryptographers tend to work in "abstract" environments
  - The real protocol is more complicated
  - COMSEC engineers don't understand the crypto well enough

OK, so what about new protocols?

- At least we don't have the problem of installed base
- Don't need to worry about compatibility
- So maybe we can do the right thing from the start

A new protocol: JavaScript Object Signing and Encryption

- Basic idea: a Web-friendly secure messaging scheme
- A number of Web settings need cryptographically secure messaging
  - OAuth, BrowserID, etc.
- Lots of secure messaging schemes (S/MIME, PGP, etc.)
  - People hate (hate hate hate) ASN.1
  - Especially bad fit for JavaScript, which does badly with binary encodings
- JavaScript Web Signatures/JavaScript Web Encryption is based on a JSON encoding
  - Can be conveniently implemented totally in JS

JavaScript Web Encryption Example

```
{"alg":"RSA1_5",
 "enc":"A256GCM",
 "iv":"__79_Pv6-fg",
 "x5t":"7noOPq-hJ1_hCnvWh6IeYI2w9Q0"}
```

JOSE-supported algorithms

- Signature
  - RSA with SHA-256, SHA-384, SHA-512 (PKCS#1 1.5)
  - ECDSA with P-246, P-384, P-521
- Integrity
  - HMAC with SHA-256, SHA-384, SHA-512
- Public Key Encryption
  - RSA (PKCS#1 1.5), RSA-OAEP
  - ECDH with static keys
- Symmetric Encryption
  - AES-128 and AES-256 in CBC and GCM modes

JOSE-supported algorithms (blue → mandatory)

- Signature
  - RSA with SHA-256, SHA-384, SHA-512 (PKCS#1 1.5)
  - ECDSA with P-246, P-384, P-521
- Integrity
  - HMAC with SHA-256, SHA-384, SHA-512
- Public Key Encryption
  - RSA (PKCS#1 1.5), RSA-OAEP
  - ECDH with static keys
- Symmetric Encryption
  - AES-128 and AES-256 in CBC and GCM modes

Why these algorithms?

I fully agree that encryption should be used only with integrity checking. In the ideal world, we would only have to use some mode like GCM, but unfortunately the support is yet to become wide spread. In most platforms, it is likely that the develop has to rely on something like CBC whose support is much more wide spread. Under this cercumstances, we would have to support CBC with integrity checks. So, doing encryption and integrity check (optional for GCM etc., and MUST for CBC) is the way to go, IMHO. To do so, instead of just using CEK, include the CMK in the header and use it to create CEK and CIK.

...

As to the support of PKCS 1.5 is concerned, again, it is the same problem as the support of GCM. The platform that only supports PKCS 1.5 is still wide spread. Of course we may use this spec to whip the providers and software vendors to build the support for a more decent algorithms but that will severely limit the adoption of the spec in the initial phase. I think we should support PKCS 1.5 as a base line which is a MUST, and strongly recommend more decent ones and put some comment in the security consideration as well.

— Nat Sakimura (draft co-author)

Finally something we can fix

- Only a relatively small number of crypto libraries
- Some (but not all) support modern algorithms
  - E.g., OpenSSL currently supports OAEP and 1.0.1 will support PSS and GCM
- Not too hard to add support to these systems
  - Many are open source anyway
  - No real compatibility concerns
- May already be too late for JOSE

What gets systems upgraded?

- Really serious security vulnerabilities
  - Clearly explainable
  - Lack of available countermeasures
- Drop in replacement
  - This is why AES was relatively easy
- Some new capability
  - This is harder than it sounds (e.g., PAKE)
- Being required by someone important
  - Harder than it sounds..

How bad is all this?

- We have bigger problems
  - Compromise of the CA infrastructure
  - Significant implementation flaws
- Countermeasure strategy has been fairly successful
  - Old primitives were actually fairly strong
- But the weight of the hacks is starting to add up
- Nice to at least avoid importing it into new systems
- Your mission, should you choose to accept it

Questions?

References

[Ble98] Daniel Bleichenbacher. Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS #1". In *Advances in Cryptology – CRYPTO'98*, volume 1462 of *LNCS*, pages 1–12, 1998.

[BR96] Mihir Bellare and Philip Rogaway. Optimal Asymmetric Encryption - How to Encrypt with RSA. In *Advances in Cryptology – Eurocrypt '96*, volume 1070, pages 399–416, 1996.

[BR06] Steven M. Bellovin and Eric Rescorla. Deploying a New Hash Algorithm. In *Proceedings of ISOC NDSS 2006*, February 2006.

[BWNH+03] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. RFC 3546, Internet Engineering Task Force, June 2003.

[Cho02] P. Chown. Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS). RFC 3268, Internet Engineering Task Force, June 2002.

[CHVV03] Brice Canvel, Alain Hiltgen, Serge Vaudenay, and Martin Vuagnoux. Password Interception in a SSL/TLS Channel. In *Advances in*

Cryptology – CRYPTO'03, 2003.

[DR06] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, Internet Engineering Task Force, April 2006.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Internet Engineering Task Force, August 2008.

[FGK03] S. Frankel, R. Glenn, and S. Kelly. The AES-CBC Cipher Algorithm and Its Use with IPsec. RFC 3602, Internet Engineering Task Force, September 2003.

[JK03] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447, Internet Engineering Task Force, February 2003.

[Kra01] Hugo Krawczyk. The Order of Encryption and Authentication for Protecting Communications (Or: How Secure is SSL?). In *Advances in Cryptology – CRYPTO'01*, 2001.

[Moe] Bodo Moeller. Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures.

http://www.openssl.org/~bodo/tls-cbc.txt.

[MR04] Nagendra Modadugu and Eric Rescorla. The Design and Implementation of Datagram TLS. In *Proceedings of ISOC NDSS 2004*, February 2004.

[NIS01] NIST. Specification for the Advanced Encryption Standard (AES), nov 2001. FIPS PUB 197.

[NIS02] NIST. Secure Hash Standard, aug 2002. FIPS PUB 180-2.

[Ris11] Ivan Ristić. State of SSL, April 2011. file:///Users/ekr/Downloads/Qualys_SSL_Labs-State_of_SSL_InfoSec_World_April_2011.pdf.

[Riv92] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, Internet Engineering Task Force, April 1992.

[Sch03] J. Schaad. Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS). RFC 3565, Internet Engineering Task Force, July 2003.

[SSA+09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA

Certificate. In *Advances in Cryptology – CRYPTO'09*, 2009.

[WMSM11] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose. Phonotactic reconstruction of encrypted VoIP conversations: Hookt on fon-iks. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, May 2011.

[YRS+09] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In Anja Feldmann and Laurent Mathy, editors, *Proceedings of IMC 2009*, pages 15–27. ACM Press, November 2009.