Randomness Extractors. Secure Communication in Practice

Lecture 17

School on MPC at IIT B!

March 27-29

(followed by a 2-day Crypto workshop)

Monday	11:00 - 12:30	**What is MPC?**	Manoj
	2:00 - 3:00	**Zero Knowledge**	Muthu
	3:30 - 5:00	**Garbled Circuits**	Arpita
Tuesday	9:00 - 10:30	**Randomized Encoding**	Yuval
	11:00 - 12:30	**Oblivious Transfer**	Arpita
	2:00 - 3:30	**Composition**	Muthu
	4:00 - 5:00	**MPC Complexity**	Manoj
Wednesday	9:00 - 10:30	**Honest-Majority MPC**	Vassilis
	11:00 - 12:30	**"MPC in the head"**	Yuval
	2:00 - 3:00	**Asynchronous MPC**	Vassilis

Yuval Ishai
Technion & UCLA

Muthu Venkitasubramaniam
U Rochester

Arpita Patra
IISc

Vassilis Zikas
RPI

Manoj Prabhakaran
IIT Bombay

Randomness Extraction

Randomness Extractors

- Consider a PRG which outputs a pseudorandom group element in some complicated group
  - A standard bit-string representation of a random group element may not be (pseudo)random
  - Can we efficiently map it to a pseudorandom bit string? Depends on the group...
- Suppose a chip for producing random bits shows some complicated dependencies/biases, but still is highly unpredictable
  - Can we purify it to extract uniform randomness? Depends on the specific dependencies...
- A general tool for purifying randomness: **Randomness Extractor**

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
  - "Optimal" in all parameters except seed length
- Constructions with shorter seeds known
  - e.g. Based on expander graphs

Randomness Extractors

- **Strong extractor**: output is random even when the seed for extraction is revealed
  - 2-UHF is an example
- Useful in key agreement
  - Alice and Bob exchange a non-uniform key, with a lot of pseudoentropy for Eve (say, g^{xy})
  - Alice sends a random seed for a strong extractor to Bob, in the clear
  - Key derivation: Alice and Bob extract a new key, which is pseudorandom (i.e., indistinguishable from a uniform bit string)

Randomness Extractors

- **Pseudorandomness Extractors** (a.k.a. computational extractors): output is guaranteed only to be pseudorandom if input has sufficient (pseudo)entropy
- Key Derivation Function: Strong pseudorandomness extractor
  - Cannot directly use a block-cipher, because pseudorandomness required even when the randomly chosen seed is public ("salt")
    - Extract-Then-Expand: Enough to extract a key for a PRF
    - Can be based on HMAC or CBC-MAC: Statistical guarantee, if compression function/block-cipher is a random function/random permutation
    - Models IPsec Key Exchange (IKE) protocol. HMAC version later standardised as HKDF.

Randomness Extractors

- Extractors for use in system Random Number Generator (think /dev/random)
  - Additional issues:
    - Online model, with a variable (and unknown) rate of entropy accumulation
    - Should recover from compromise due to low entropy phases
  - Constructions provably secure in such models known
    - Using PRG (e.g., AES in CTR mode), universal hashing and "pool scheduling" (similar to Fortuna, used in Windows)

Secure Communication In Practice

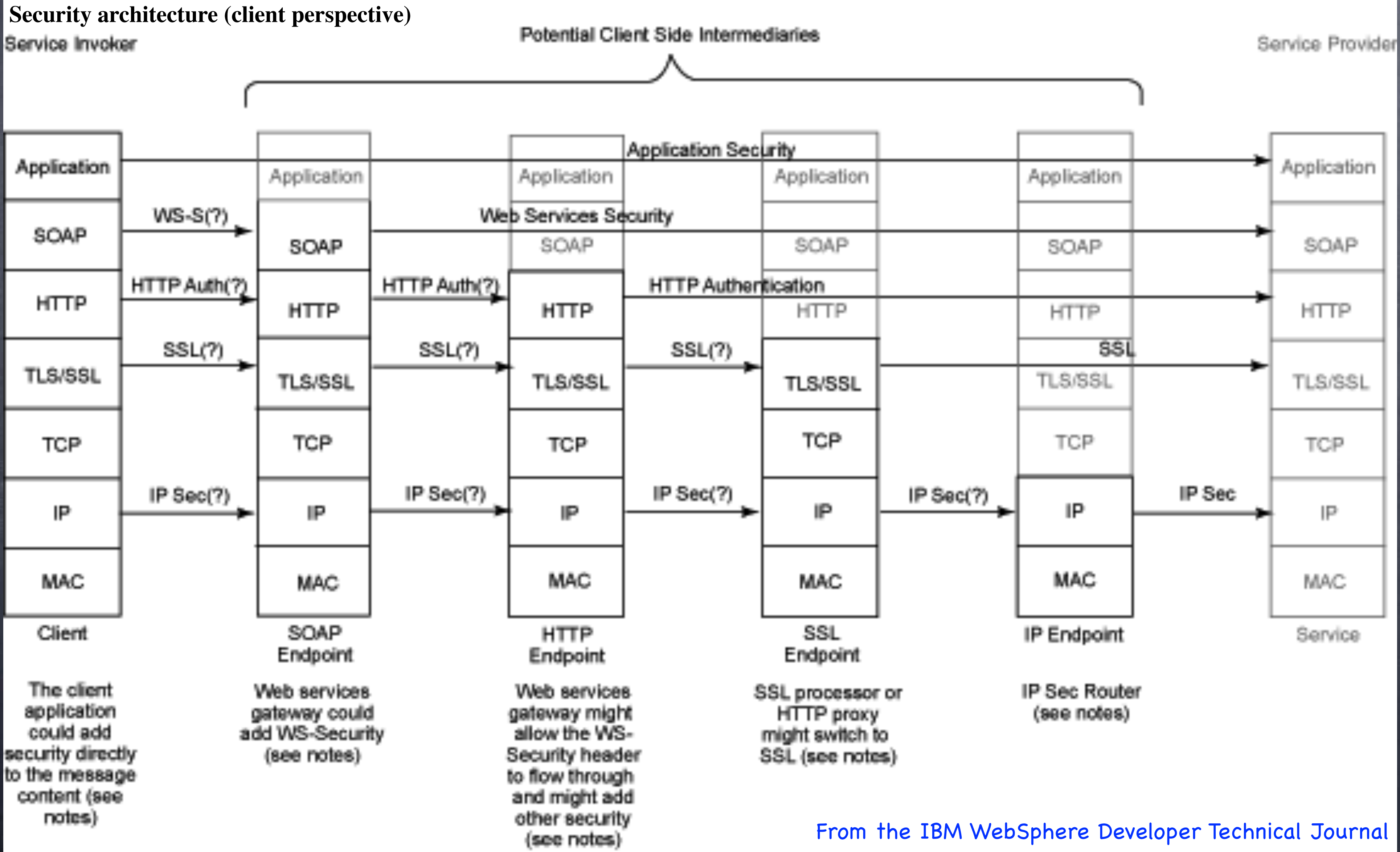
We saw...

- Symmetric-Key Components
  - SKE, MAC
- Public-Key Components
  - PKE, Digital Signatures
- Building blocks: Block-ciphers (AES), Hash-functions (SHA-3), Trapdoor PRG/OWP for PKE (e.g., DDH, RSA) and Random Oracle heuristics (in RSA-OAEP, RSA-PSS)
- Symmetric-Key primitives much faster than Public-Key ones
  - Hybrid Encryption gets best of both worlds

Secure Communication in Practice

- Can do at application-level
  - e.g. between web-browser and web-server
- Or lower-level infrastructure to allow use by more applications
  - e.g. between OS kernels, or between network gateways
- Standards in either case
  - To be interoperable
  - To not insert bugs by doing crypto engineering oneself
  - e.g.: SSL/TLS (used in https), IPSec (in the "network layer")

Security Architectures
(An example)



From the IBM WebSphere Developer Technical Journal

Secure Communication Infrastructure

- Goal: a way for Alice and Bob to get a private and authenticated communication channel (can give a detailed SIM-definition)
- Simplest idea: Use a (SIM-CCA secure) public-key encryption (possibly a hybrid encryption) to send signed (using an existentially unforgeable signature scheme) messages (with sequence numbers and channel id)
    - Limitation: Alice, Bob need to know each other's public-keys
- But typically Alice and Bob engage in "transactions," exchanging multiple messages, maintaining state throughout the transaction
    - Makes several efficiency improvements possible

Secure Communication Infrastructure

- Secure Communication Sessions
  - Handshake protocol: establish private shared keys
  - Record protocol: use efficient symmetric-key schemes
- Server-to-server communication: Both parties have (certified) public-keys
- Client-server communication: server has (certified) public-keys
  - Client "knows" server. Server willing to talk to all clients
- Client-Client communication (e.g., email) Clients share public-keys in ad hoc ways

(Authenticated) Key-Exchange

Server may "know" (some) clients too, using passwords, pre-shared keys, or if they have (certified) public-keys. Often implemented in application-layer

Certificate Authorities

- How does a client know a server's public-key?
  - Based on what is received during a first session? (e.g., first ssh connection to a server)
- Better idea: Chain of trust
  - Client knows a certifying authority's public key (for signature)

COMODO Creating Trust Online®

Symantec

Go Daddy®

Google Trust Services

Certificate Authorities

- How does a client know a server's public-key?
  - Based on what is received during a first session? (e.g., first ssh connection to a server)
- Better idea: Chain of trust
  - Client knows a certifying authority's public key (for signature)
  - Bundled with the software/hardware
- Certifying Authority signs the signature PK of the server
  - CA is assumed to have verified that the PK was generated by the "correct" server before signing
  - Validation standards: Domain/Extended validation

Forward Secrecy

- Servers have long term public keys that are certified
  - Would be enough to have long term signature keys, but in practice long term encryption keys too
  - Problem: if the long term key is leaked, old communications are also revealed
    - Adversary may have already stored, or even actively participated in old sessions
  - Solution: Use fresh public-keys/do a fresh key-exchange for each session (authenticated using signatures)

A Simple Secure Communication Scheme

- Handshake
  - Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)
- For authentication only: use MAC
  - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
- For authentication + (CCA secure) encryption: encrypt-then-MAC
  - stream-cipher, and "stream-MAC"

Server's PK either trusted (from a previous session for e.g) or certified by a trusted CA, using a Digital Signature scheme

Need to avoid replay attacks (infeasible for server to explicitly check for replayed ciphertexts)

Recall "inefficient" domain-extension of MAC: Add a session-specific nonce and a sequence number to each message before MAC'ing

Authentication for free: MAC serves dual purposes!

TLS (SSL)

- Handshake
  - Client sends session keys for MAC and SKE to the server using SIM-CCA secure PKE, with server's PK (i.e. over an unauthenticated, but private channel)
- For authentication only: use MAC
  - In fact, a "stream-MAC": To send more than one message, but without allowing reordering
- For authentication + (CCA secure) encryption: encrypt-then-MAC
  - stream-cipher, and "stream-MAC"

Negotiations on protocol version etc. and "cipher suites" (i.e., which PKE/key-exchange, SKE, MAC (and CRHF)).

e.g. cipher-suite: RSA-OAEP for key-exchange, AES for SKE, HMAC-SHA256 for MAC

Server sends a certificate of its PKE public-key, which the client verifies

Server also "contributes" to key-generation (to avoid replay attack issues): Roughly, client sends a key K for a PRF; a master key generated as $PRF_K(x,y)$ where x from client and y from server. SKE and MAC keys derived from master key

Uses MAC-then-encrypt! Not CCA secure in general, but secure with stream-cipher (and with some other modes of block-ciphers, like CBC)

Several details on closing sessions, session caching, resuming sessions ...

TLS: Some Considerations

- Overall security goal: Authenticated and Confidential Channel Establishment (ACCE), or Server-only ACCE
- Handshake Protocol
  - Cipher suites are negotiated, not fixed → "Downgrade attacks"
  - Doesn't use CCA secure PKE, but overall CCA secure if error in decryption "never revealed" (tricky to ensure!)
- Record Protocol
  - Using MAC-then-Encrypt is tricky:
    - CCA-secure when using SKE implemented using a stream cipher (or block-cipher in CTR mode) or CBC-MAC
    - But insecure if it reveals information from decryption phase.
      - e.g., different times taken by MAC check (or different error messages!) when a format error in decrypted message

TLS: Some Considerations

- Numerous vulnerabilities keep surfacing

  FREAK, DROWN, POODLE, Heartbleed, Logjam, …

  And numerous unnamed ones: www.openssl.org/news/vulnerabilities.html

  Listed as part of Common Vulnerabilities and Exposures (CVE) list: cve.mitre.org/
- Bugs in protocols
  - Often in complex mechanisms created for efficiency
  - Often facilitated by the existence of weakened "export grade" encryption and improved computational resources
  - Also because of weaker legacy encryption schemes (e.g. Encryption from RSA PKCS#1 v1.5 — known to be not CCA secure and replaced in 1998 — is still used in TLS)
- Bugs in implementations
- Side-channels originally not considered
- Back-Doors (?) in the primitives used in the standards

TLS: Some Considerations

- Numerous vulnerabilities keep surfacing
  - FREAK, DROWN, POODLE, Heartbleed, Logjam, …
  - And numerous unnamed ones: www.openssl.org/news/vulnerabilities.html
  - Listed as part of Common Vulnerabilities and Exposures (CVE) list: cve.mitre.org/

(Kenny Paterson & Thyla van der Merwe, Dec 2016)

Encryption from RSA PKCS#1 v1.5 — known to be not CCA secure and replaced in 1998 — is still used in TLS)

- Bugs in implementations
- Side-channels originally not considered
- Back-Doors (?) in the primitives used in the standards

Beyond Communication

- Encryption/Authentication used for data at rest
  - e.g., disk encryption, storing encrypted data on a cloud server, ...
- Security definitions like SIM-CCA do not directly extend to all these settings
  - New concerns that do not arise in setting up communication channels
  - e.g., circular (in)security: encrypting the SK using its own PK