Randomness

Lecture 14

Randomness Extractors
Key Derivation Functions

Randomness is Key!

- Security of all cryptographic primitives depends on the assumption that random samples can be drawn
  - Typically random bit strings (used as keys for block ciphers, or converted to prime numbers, group elements, etc. for key generation in public-key cryptography)
- If this assumption is violated, serious security issues
- A study in 2012 found too many RSA modulii (N=PQ) on the web shared a common factor (as revealed by GCD computation)
- In libssl, a poor source of randomness was used. In particular, in Debian's version from September 2006 to May 2008, it was easily predictable.
  - Seriously undermines security. E.g., users with certificates generated in such systems could be spoofed until certificates expired/were revoked

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

- Takes an input with high unpredictability, and an independent seed as a "catalyst", and outputs an (almost) random string
- Statistical guarantees: output not just pseudorandom, but statistically (almost) uniform, if input has sufficient entropy
- 2-Universal Hash Functions (when sufficiently compressing), where the seed is the hash function
  - "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 in fact a strong extractor (seed is the hash function)
  - "Left-Over Hash Lemma"
- 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

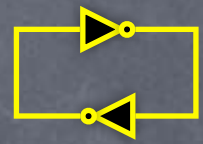
- **Pseudorandomness Extractors** (a.k.a. computational extractors): output is guaranteed only to be pseudorandom if input has sufficient (pseudo)entropy
- Key Derivation Function: Strong pseudorandomness extractor
  - Cannot directly use a block-cipher, because pseudorandomness required even when the randomly chosen seed is public ("salt")
    - Extract-Then-Expand: It's enough to extract a key for a PRF
    - Can be based on HMAC or CBC-MAC: Statistical guarantee, if compression function/block-cipher were a public but randomly chosen function/permutation
    - Models KDF in IPsec's Internet 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 (especially in the beginning)
  - 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)

Randomness Hardware

- Originally, analog circuitry for amplifying thermal noise. But many drawbacks in a digital processor.
- Nowadays digital circuity as entropy source



  - Since the component NOT gates are not identical, circuit needs to be dynamically controlled to make each bit unbiased
  - Switching on and off expected to make the bits independent
- RDRand instruction in Intel and AMD processors
  - Bits from the digital entropy source are first processed using an (ad hoc) extractor with a fixed seed
  - Then used as the seed for a fast PRG (AES in CTR mode)

Randomness Reuse

- Various cryptographic schemes require that randomness is not reused
  - E.g., IV in a CPA-secure encryption
  - Randomness in Schnorr signature (Recall special soundness: responses to two challenges given the same initial message allows extraction of the signing key)
- But attacks/accidents may force randomness reuse
  - E.g., two instances of a virtual machine starting from the same snapshot
- Sometimes can mitigate the effect of randomness reuse
  - Synthetic IV: Only effect of using the same IV many times would be to reveal which messages are equal to which
    - E.g., GCM-SIV mode

Randomness Leakage

- We expect n-bit random strings (nonces, keys, etc.) to have n bits of entropy (i.e., uniform over all 2^n possibilities) even conditioned on what an adversary knows
  - Or be indistinguishable from that
- In particular, the random strings should not be leaked outside the cryptographic process which requests it
- But often side channels exist
  - E.g., CrossTalk in intel processors (revealed in June 2020): The output of RDRand (among other things) was stored in a staging buffer, which processes in other cores could access
    - Leads to extraction of signing keys from within "secure enclaves" provided by Intel SGX