Wrap Up: Cryptographic Primitives

Lecture 15

Alternate Assumptions for PKE
Randomness Extractors

Story So Far

- Basic primitives for secure communication:

	Shared-Key	Public-Key
Encryption	SKE	PKE
Authentication	MAC	Signature

- In principle, OWF/OWP sufficient for SKE, MAC and Digital Signatures. PKE needs more structure (e.g., Trapdoor OWP)
- In practice, SKE and MAC based on ad hoc constructions of PRFs and CRHFs. Digital Signatures as well as PKE (together, public-key cryptography) uses mathematical structure, and often relies on the random oracle model.

PKE Maths

- Initially PKE was based on hardness of problems in modular arithmetic (RSA/factoring, modular discrete log)
- Problems from several other areas, since then
  - Elliptic curve cryptography (mainstream, currently)
  - Code-based crypto
  - Lattice-based crypto
  - Multivariate Polynomial crypto

"Post-Quantum Crypto" candidates

Elliptic Curve Crypto

- Starting 1985 (by Miller, Koblitz)
- Groups where Discrete log (and DDH) is considered much harder than in modular arithmetic, and hence much smaller groups can be used.
- Given a finite field F, one can define a commutative group $G \subseteq F^2$, as points (x,y) which lie on an "elliptic curve," with an appropriately defined group operation
  - Different curves yield different groups
- Today, most popular PKE schemes use Diffie-Hellman over elliptic curves specified by various standards.
  - Pro: Significantly faster than the other options!
  - Con: Which elliptic curves are good?

Code-Based Crypto

- Coding theory based, since McEliece crypto system (1978)
  - A linear code is specified by a matrix G. Message x is encoded into a codeword xG. Can easily check if c is a codeword.
  - Structured linear codes exist for which error correcting algorithms can correct sparse errors — i.e., recover x from xG+e where the error vector e has a large fraction of 0s
  - But for a random linear code, this seems hard
  - Idea: Masquerade structured codes to look random. Secret key reveals the original structured code. Encrypt as a codeword plus a sparse noise vector.
- Not commonly used today, as large key sizes and slow computation

Code-Based Crypto

- G: a k × n generator matrix for a good code over a GF(2)
- S: a random k × k invertible matrix
- P: a random n × n permutation matrix
- Public Key: H = SGP, private key = (S,G,P)
- Encryption: mH+e, where e is a random sparse vector (sparse enough to allow error correction for the original code)
- Decryption: Let d := cP^{-1} = mSG+e', where e'=eP^{-1} as sparse as e. Recover m := Decode(d)·S^{-1}
- Not CPA secure! [Why?] Can check if c-mH is sparse
- Use [r m] instead of m, r being a random pad
  - CPA secure under the assumptions that H is pseudorandom and "Learning Parity with Noise" is hard for random H

Lattice-Based Crypto

- Lattice: set of (real) vectors obtained by linear combination of basis vectors using only integer coefficients
  - Hard problems related to finding short vectors in the lattice
- Original use of lattices: to break a candidate for PKE (called the "Knapsack cryptosystem") by Merkle and Hellman
- Constructions: NTRU (1996), Ajtai/Ajtai-Dwork (1996/97), ...
- More recent constructions based on the Learning With Errors (LWE) assumption over $\mathbb{Z}_q$.
  - (A, Ax + e) is pseudorandom when e is a "short" noise vector
  - Known to hold if some lattice problems are hard

Lattice-Based Crypto: PKE

- NTRU approach: Private key is a "good" basis, and the public key is a "bad basis"
  - Worst basis (one that can be efficiently computed from any basis): Hermite Normal Form (HNF) basis
- To encrypt a message, encode it (randomized) as a short "noise vector" v. Output c = v+u for a random lattice point u that is chosen using the public basis
  - To decrypt, use the good basis to find u as the closest lattice vector to c, and recover v = c-u
- NTRU Encryption: use lattices with succinct basis
- Conjectured to be CPA secure for appropriate lattices. No security reduction known to simple lattice problems

Lattice-Based Crypto: PKE

cf. El Gamal: $A \to g$, $S \to y$, $P \to Y = g^y$

$a \to x$, $u \to g^x$, $a^T P \to Y^x$

- An LWE based approach:
  - Public-key is (A,P) where P=AS+E, for random matrices (of appropriate dimensions) A and S, and a noise matrix E over $\mathbb{Z}_q$
  - To encrypt an n bit message, map it to an ("error-correctable") vector v; pick a random "noise vector" a (i.e., small coordinates); ciphertext is (u,c) where $u^T = a^T A$ and $c^T = a^T P + v^T$
  - Decryption using S: recover message from $c^T - u^T S = v^T + a^T E$, by "error correcting" (error not sparse, but has small entries)
  - CPA security: By LWE assumption, P in the public-key is indistinguishable from random; and, encryption under truly random (A,P) loses essentially all information about the message
    - Coming up: $a^T P$ acts as a one-time pad, even given A, P, $a^T A$

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 <u>uniform</u> 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)
  - Needs a seed, chosen independent of the input (aka source)
- 2-Universal Hash Functions (when sufficiently compressing) are extractors
  - The seed specifies the hash function h from the family: h(x) is almost uniform, when x has high entropy, and h is chosen from the 2UHF family uniformly and independently of x.
  - "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: (h,h(x)) is almost uniform
- 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)
- In LWE-based PKE
  - $h_M(x) = Mx$, where M compressing, $x \neq 0$, is a 2-UHF [Exercise]
  - a (even with small entries) has enough entropy given (A, A^Ta), and so P^Ta almost uniform even given (A, P, A^Ta)

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
  - Constructions provably secure in such models known
    - Using PRG, universal hashing and "pool scheduling" (similar to Fortuna, used in Windows)

Coming Up

- Secure communication in practice
  - SSL/TLS
  - IPSec
  - BGPSec
  - DNSSec