# Layer 1 – Low-Level Primitives

**Libraries to use:** Bouncy Castle, CACE, Crypto++

**General**

1. GetRandom: use cryptographic call, if possible from OS
2. Randomness extraction HKDF: <http://eprint.iacr.org/2010/264>

**Pseudorandom generator**

1. RC4 (throw out first 1024 bytes)
2. SHA-based PRG

**Pseudorandom permutations**

1. AES\_128
2. AES\_256
3. 3DES
4. PRP for arbitrary input/output length [Katz-Lindell, Section 6.6; start with PRF with length half of that needed for the PRP]

**Pseudorandom functions**

1. All of the pseudorandom permutations
2. HMAC with any hash function
3. PRFs with arbitrary input and output length
   1. Arbitrary input length: using any of the HMAC-SHA constructions
   2. Arbitrary output length:
      1. If shorter than basic, truncate
      2. If longer than basic: apply multiple times with CTR index (if do this, then the concrete instance with a given key must ALWAYS use this mode)

**Hash functions**

1. SHA-1
2. SHA-256
3. SHA-224 (this is truncated SHA-256)
4. SHA-512
5. SHA-384 (this is truncated SHA-512)
6. Universal one-way/target collision-resistant hashing: <http://webee.technion.ac.il/~hugo/rhash/rhash.pdf>
7. Perfect universal hash functions (use package giving GF[2k] computations)
8. GF[2k] operations: <http://www.shoup.net/ntl/doc/tour-modules.html>
9. <http://www.cosic.esat.kuleuven.be/publications/article-73.ps>

**Trapdoor permutations**

1. RSA\_1024
2. RSA\_2048
3. RSA\_4096
4. Rabin\_1024 [Katz-Lindell, Section 11.2]
5. Rabin\_2048 [Katz-Lindell, Section 11.2]

**Discrete log**

1. **Zp\*** for p of length 1024, with p=2q+1 and q prime (group of order q); fine also with general q
2. **Zp\*** for p of length 2048, with p=2q+1 and q prime (group of order q)
3. Elliptic curve groups: as given by NIST and implemented in bouncy castle, CACE or whatever you find

Minimal operations:

1. Get generator
2. Get random element
3. Multiply group elements
4. Add and multiply mod q
5. Compute inverse of a group element
6. Exponentiate
7. Multiple exponentiations with same base
8. grhs computations
9. ***Bilinear operations*** [references, CACE]

**Quadratic residuosity (QR)** (version 2.0 of the SDK; not at this stage)

1. Modulus of length 1024
2. Modulus of length 2048

**Lattice constructions** (version 2.0)

**Note on lengths:** For all of the above, if the underlying library allows more flexibility with sizes (modulus size and dlog with arbitrary sizes for p and q) then this is even better. The above is the “minimum”.

# Layer 2 – Non-Interactive Primitives

**Message authentication codes**

1. CBC-MAC with length in first block, using all pseudorandom permutations
2. HMAC for any hash function

**Symmetric encryption**

1. CBC mode (random IV) using all pseudorandom permutations
2. CTR mode (random IV) using all pseudorandom permutations
3. Encrypt-then-MAC (any encrypt mode and any MAC)

**Asymmetric encryption**

1. El Gamal over given discrete log groups
2. RSA-OAEP, PKCS v2.1
3. Cramer-Shoup – DDH (<http://en.wikipedia.org/wiki/Cramer%E2%80%93Shoup_cryptosystem>)
4. Cramer-Shoup – N-residuosity (version 2.0)
5. Paillier\_1024 (Damgard-Jurik version) [BENNY]
6. Paillier\_2048 (Damgard-Jurik version) [BENNY]
7. QR\_Blum-Goldwasser (version 2.0)

**Homomorphic encryption**

1. Paillier (Damgard-Jurik version) [BENNY]
2. El Gamal (in the exponent)
3. QR\_Blum-Goldwasser (version 2.0)

**Digital signatures**

1. RSA-PSS, PKCS v2.1
2. DSA, ECDSA
3. Hash-based one-time signatures [Katz-Lindell, Section 12.5]
4. Cramer-Shoup (version 2.0)

# Layer 3 – Interactive Protocols

**Standard commitment schemes**

1. Pedersen commitments: <http://cs.nyu.edu/courses/fall01/G22.3033-003/lect/lecture14.ps> , Section 2.5
2. Hash-based (random-oracle) commitments: define Commit(x) = HASH(x||r) where r is 128-bits random
3. Hash-based commitments: <http://cs.nyu.edu/courses/fall01/G22.3033-003/lect/lecture14.ps> , Section 2.3
4. Public-key encryption based commitments: commit to x by choosing new (pk,sk) and sending (pk,E(pk,x))
5. UC-secure commitment <http://eprint.iacr.org/2001/091>

**Trapdoor (equivocal) commitment schemes**

1. Based on DLOG Sigma protocol [HL, Section 6.6, use DLOG sigma as basis]
2. General transformation from Sigma protocol where simulator instructions are provided.

**Extractable commitment schemes**

1. Any commitment with a ZKPOK of committed value
2. Others?

**Homomorphic commitment schemes**

1. Take from LEGO: <http://eprint.iacr.org/2008/427.pdf>

**Sigma protocol**

1. Sigma protocol of DLOG [HL, Section 6.1]
2. Sigma protocol of DH tuple [HL, Section 6.2]
3. Sigma protocols for Damgard-Jurik [BENNY]
4. Sigma of committed value for Pedersen commitments [???]
5. Sigma of committed value for ElGamal commitments [???]
6. Sigma that committed value is as given - Pedersen [???]
7. Sigma that committed value is as given - El Gamal [???]
8. Template for Sigma protocol – programmer fills in procedures as below and Sigma protocol is built automatically:
   1. Prover compute 1st message
   2. Prover compute 2nd message
   3. Verifier check
   4. Verifier query length
   5. Optional: check membership in LR. That is, given x it is easy to decide if there

exists a w such that (x,w) ∈ R

* 1. Optional: simulator instructions for generating prover messages (for OR, general compound statements and trapdoor commitments)

1. AND of any number of Sigma protocols [HL, Section 6.4]
2. OR of any two Sigma protocols [HL, Section 6.4]
3. General compound statements [BENNY]

**Zero knowledge**

1. Zero-knowledge for every Sigma-protocol using any commitment [HL, Section 6.5.1]
2. ZKPOK for every Sigma-protocol using any trapdoor commitment [HL, Section 6.5.2]
3. Fiat-Shamir transform for any Sigma protocol: just get verifier message by HASH(x,\alpha)

**Coin tossing**

1. Basic Blum single-coin tossing using any commitment scheme
   1. P1 commits to a single random bit using any commitment scheme
   2. P2 sends a random bit to P1
   3. P1 decommits
   4. Both parties output XOR of bits
2. [Lindell01] coin tossing, using Pedersen commitments and DLOG-ZK
   1. P1 commits to a random r using Pedersen
   2. P1 proves in ZKPOK that it knows the committed value (item 4 in sigma)
   3. P2 sends a random s
   4. P1 sends r (without decommitting)
   5. P1 proves in ZKPOK that r is the committed value (item 6 in sigma)
   6. Both parties output XOR of r and s
3. Semi-simulatable coin-tossing
   1. P1 sends a perfectly-hiding commitment to r (e.g. Pedersen or random-oracle)
   2. P2 sends a perfectly-binding commitment to s (e.g., Public-key commit or random-oracle)
   3. P1 opens
   4. P2 opens
   5. Both parties output XOR of r and s

**Oblivious transfer**

1. Naor-Pinkas (using any DH group) [HL, Section 7.2.1]
2. AIR (using any homomorphic encryption) [HL, Section 7.2.2]
3. HL-one-sided (using any DH group) [HL, Section 7.3]
4. ~~HL-full simulation (using any DH group) [HL, Section 7.4]~~
5. PVW\_plain (using any DH group or N-residuosity) [HL, Section 7.5] and [PVW]
6. PVW\_plain (using a Fiat-Shamir proof instead) [HL, Section 7.5 at the end]
7. PVW\_UC (using any DH group or N-residuosity) [PVW], <http://www.cc.gatech.edu/~cpeikert/pubs/OTpaper.pdf>

**Batch OT**

1. ~~HL-full-sim [HL, Section 7.4.2]~~
2. PVW-batch [HL, Section 7.5]

**Oblivious polynomial evaluation**

1. Based on OT [BENNY]
2. Based on homomorphic encryption [BENNY]

**Oblivious pseudorandom function evaluation**

1. Based on OT [HL, Section 7.6]

**Authenticated communication channels**: use openSSL

**Private communication channels**: use openSSL

**Authenticated broadcast channel**

As in document by Meital

**Information theoretic techniques** (take specification from VIFF)

1. Secret sharing (including general polynomial interpolation)
   1. Class that can work over any field
   2. Use Zp\* as basic field (here p can be small)
2. VSS
3. Arithmetic circuit protocols (addition, multiplication…)

# Important Note

A crucial part of the design phase will be a pseudocode write-up of all the required primitives. This is to ensure that the translation from a protocol described in a paper to one that is to be implemented is correct.

## Bibliography

Katz-Lindell: Introduction to Modern Cryptography

HL: Hazay Lindell, Efficient Secure Two-Party Computation