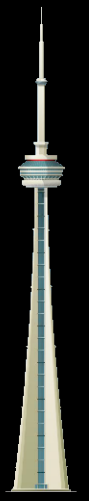
Hash Based Signatures Update

draft-mcgrew-hash-based-sigs-02

July 2014 – IETF 90 Toronto

David McGrew
mcgrew@cisco.com

Michael Curcio
micurcio@cisco.com



Normative Changes

- Added SHA-512 based algorithms
  - Needed for postquantum security
  - Exercises ‘algorithm agility’
- MTS public key contains mts_algorithm_type
- MTS private key format added
- Type values are now byte palindromes

Palindromes eliminate htonl/ntohl

```
enum mts_algorithm_type {
   mts_reserved       = 0x00000000,
   mts_sha256_k2_h20 = 0x01000001,
   mts_sha256_k4_h10 = 0x02000002,
   mts_sha256_k8_h7  = 0x03000003,
   mts_sha256_k16_h5 = 0x04000004,
   mts_sha512_k2_h20 = 0x05000005,
   mts_sha512_k4_h10 = 0x06000006,
   mts_sha512_k8_h7  = 0x07000007,
   mts_sha512_k16_h5 = 0x08000008
 };
```

OTS Algorithms

Signature Algorithm	Hash	Bytes in Signature	Key Gen Time (ms)
LDWM_SHA512_M64_W1	SHA-512	16964	-
LDWM_SHA512_M64_W2	SHA-512	8516	-
LDWM_SHA512_M64_W4	SHA-512	4292	-
LDWM_SHA512_M64_W8	SHA-512	2180	-
LDWM_SHA256_M32_W1	SHA-256	8480	0.54
LDWM_SHA256_M32_W2	SHA-256	4256	0.35
LDWM_SHA256_M32_W4	SHA-256	2144	0.42
LDWM_SHA256_M32_W8	SHA-256	1088	2.56
LDWM_SHA256_M20_W1	SHA-256	5300	0.54
LDWM_SHA256_M20_W2	SHA-256	2660	0.35
LDWM_SHA256_M20_W4	SHA-256	1340	0.42
LDWM_SHA256_M20_W8	SHA-256	680	2.56

MTS Algorithms

Signature Algorithm	Hash	Bytes in Signature
MTS_SHA512_K2_H20	SHA-512	1280
MTS_SHA512_K4_H10	SHA-512	1920
MTS_SHA512_K8_H7	SHA-512	3136
MTS_SHA512_K16_H5	SHA-512	4800
MTS_SHA256_K2_H20	SHA-256	640
MTS_SHA256_K4_H10	SHA-256	960
MTS_SHA256_K8_H7	SHA-256	1568
MTS_SHA256_K16_H5	SHA-256	2400

Other Changes

- Removed some unused definitions
- Fixed several nits, clarified several points
  - Thanks are due to Andreas Hulsing, Burt Kaliski, Eric Osterweil, Ahmed Kosba, and Russ Housley
- Extended security considerations

What needs to be done

- Signature algorithm
  - Could be translated from C code
- (Optional) pseudorandom key expansion
  - Could be translated from C code
- Improved security statement
- Add ‘node label’ to hash input?
- OTS Chain?
- Multilevel MTS?

Implementation

```
[prompt]$ ./ots_driver one-time signatures test driver

USAGE
   ots_driver -c [ -k <keyprefix> ] creates a public/private keypair
   ots_driver -s <file> [ -k <keyfile> ] signs <file> with private <keyfile>
   ots_driver -v <file> [ -k <keyfile> ] [ -f <sigfile> ] verifies <file>
                with public <keyfile> and signature <sigfile>
   ots_driver -h <hexstring> [ -k <keyfile> ] signs <hexstring> with
private <keyfile>
   ots_driver -l list available algorithms   ./ots_driver -t run timing tests
   ots_driver -p run validation tests
   ots_driver -r -f <file> read and describe signature or public or private key

OPTIONS
   -o <ots algorithm> forces the use of a particular OTS algorithm
   -d output debugging information; this option can be used in
      conjunction with the -c, -s, -v, -h, -t, -v, and -r options
```

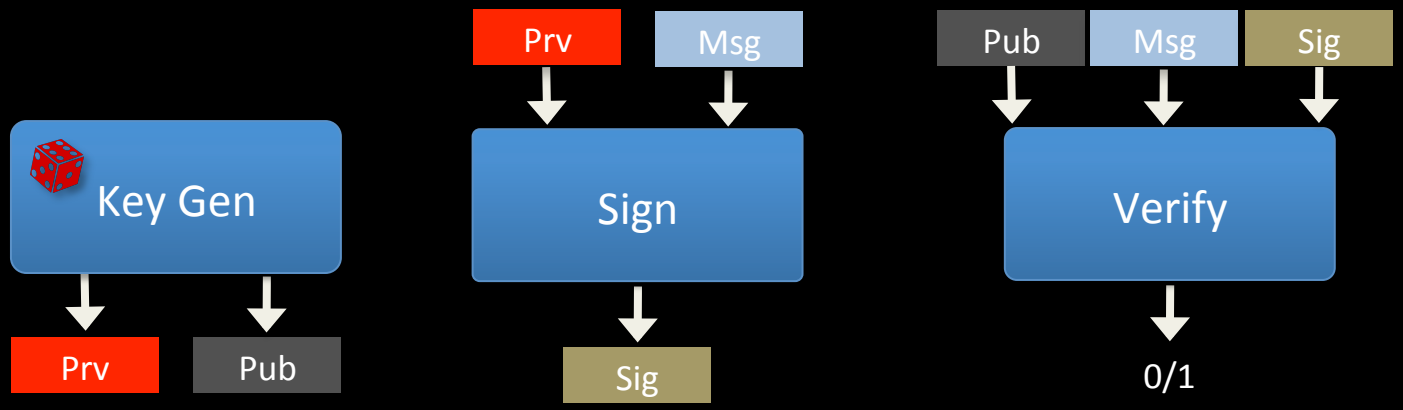
Questions and Feedback

Background on the Hash Based Signatures draft

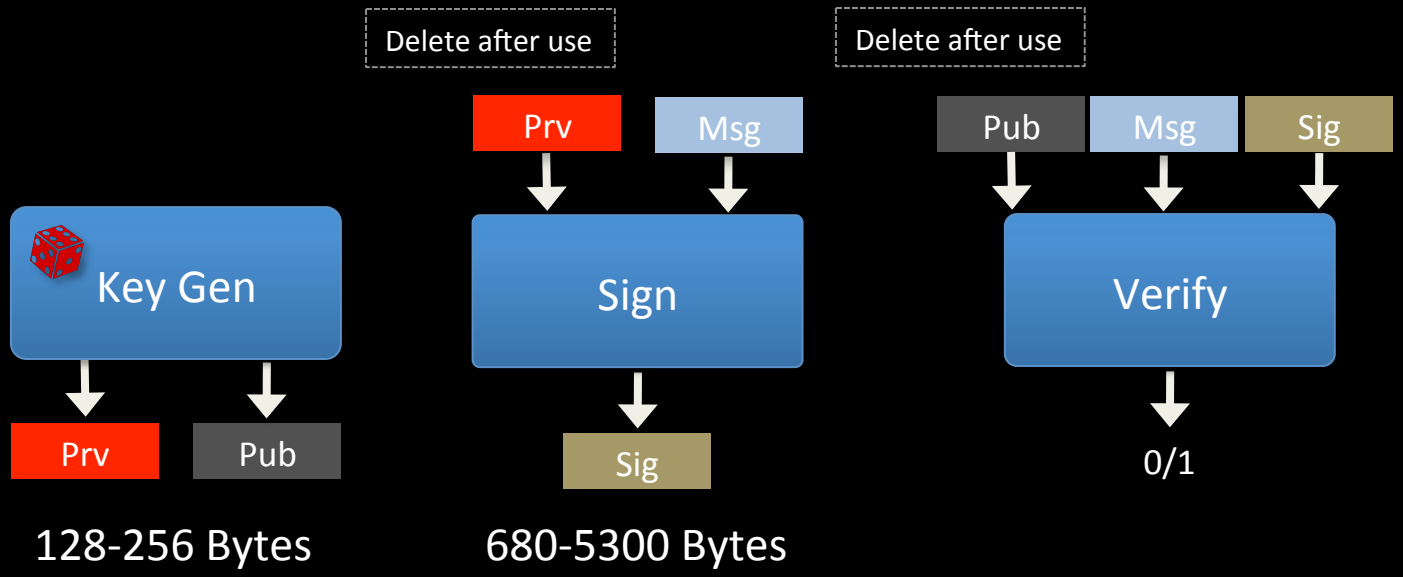
Motivations

- Compact public key signature implementations for tamper resistant modules
  - Compact verification often sufficient
  - FPGA, firmware, bootloader, OS kernel
- Security
  - Side channel resistance
  - Minimum number of security assumptions
  - Postquantum

Signature System

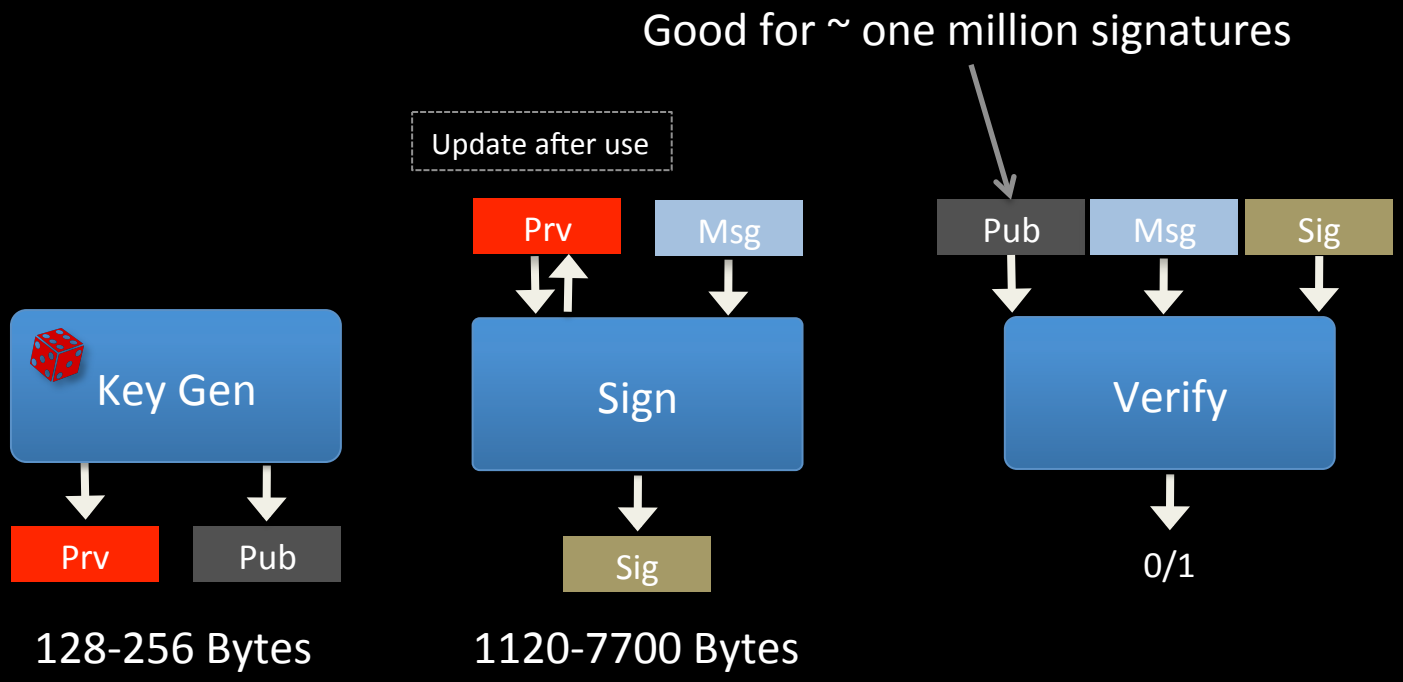


One Time Signature System



Lamport, Diffie, Winternitz, Merkle (LDWM)
[Merkle79] with checksum [Merkle87][Merkle89a][Merkle89b]

Merkle Tree Signature System



Merkle Tree Signatures [Merkle79]

Approach

- Use well established and simple methods
- Use simple data formats
  - Compact implementations can avoid ASN.1, XML
- Single identifier specifies all parameters
- Fully specify Signature Verification
- Allow signer to use different implementation strategies

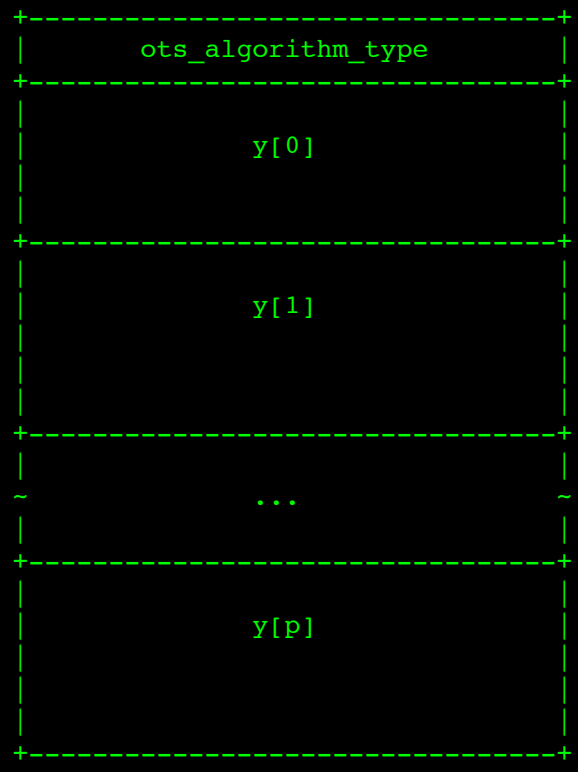
LDWM Signature Format

XDR specification

```
   enum ots_algorithm_type {
     ots_reserved        = 0,
     ldwm_sha256_m20_w1  = 0x01000001,
     ...
};

   union ots_signature switch (ots_algorithm_type type) {
    case ldwm_sha256_m32_w1:
         bytestring32 y_m32_p265[265];
    case ldwm_sha256_m32_w2:
         bytestring32 y_m3_p133[133];
    case ldwm_sha256_m32_w4:
         bytestring32 y_m32_y_p67[67];
    case ldwm_sha256_m32_w8:
         bytestring32 y_m32_p34[34];
    case ldwm_sha512_m64_w1:
         bytestring64 y_m64_p265[265];
    case ldwm_sha512_m64_w2:
         bytestring64 y_m64_p133[133];
    case ldwm_sha512_m64_w4:
         bytestring64 y_m64_y_p67[67];
    case ldwm_sha512_m64_w8:
         bytestring64 y_m64_p34[34];
    default:
      void;   /* error condition */
   };
```

Illustration



MTS Signature Format

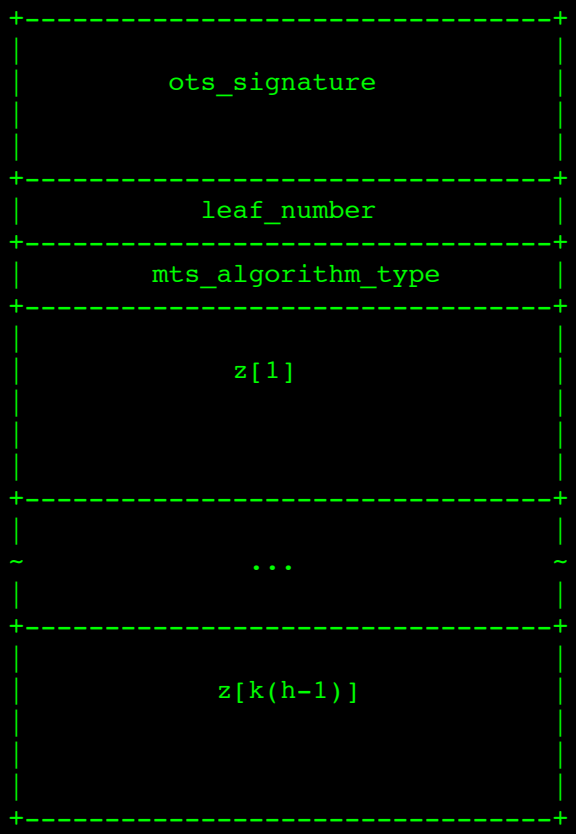
XDR specification

```
enum mts_algorithm_type {
    …
  };

  union mts_path switch (mts_algorithm_type type) {
   case mts_sha256_k2_h20:
     b32 t20[20];
   case mts_sha256_k4_h10:
     b32 t30[30];
   case mts_sha256_k8_h7:
     b32 t49[49];
   case mts_sha256_k16_h5:
     b32 t75[75];
   default:
     void;     /* error condition */
  };

  struct mts_signature_t {
    ots_signature ots_sig;
    unsigned int signature_leaf_number;
    mts_path nodes;
  };
```

Illustration



Use in standards

- draft-housley-cms-mts-hash-sig-01, *Use of the Hash-based Merkle Tree Signature (MTS) Algorithm in the Cryptographic Message Syntax (CMS)*
  - Applicable to RFC 4108, *Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages*
- Open PGP?
  - Could be based on RFC 6637 (ECC in OpenPGP)

Security Considerations

- [BDM08] Buchmann, J., Dahmen, E., and M. Szydlo, "Hash-based Digital Signature Schemes", Technische Universitat, *Darmstadt Technical Report*, 2008. https://www.cdc.informatik.tudarmstadt.de/~dahmen/papers/hashbasedcrypto.pdf
- [C:Dods05] Dods, C., Smart, N., and M. Stam, "Hash Based Digital Signature Schemes", Lecture Notes in Computer Science vol. 3796, *Cryptography and Coding*, 2005.

Pseudorandom private key expansion

- ctr_kdf_sp800_108() implements the hash/hmac based counter mode key derivation function, or CTR KDF, as specified in NIST Special Publication 800-108, Section 5.1, such that:
  - the counter field "i" is a four-byte unsigned integer
  - the Label field consists of a one-byte value indicating the purpose of the derived keying material (0xf0 is for LDWM, and 0x40 is for MTS) followed by a seven-byte unsigned integer that indicates the element number for LDWM, and indicates the leaf number for MTS.
  - the Context field is empty, i.e. a zero-length string
  - the field "L" that specifies the length, in bits, of the derived keying material is a four-byte unsigned integer. In our case, it is always equal to 256 for SHA-256.

Open Source

- POSIX LDWM and MTS implementation
  - Not yet published
- Possible applications
  - CMS integration in OpenSSL
  - Linux signed kernel modules
  - GRUB
  - OpenPGP/GPG