Update on Randomized Hashing

Shai Halevi, Hugo Krawczyk
IBM Research

http://www.ee.technion.ac.il/~hugo/rhash/
Reminder: randomized hashing

- To hash a message m:
  - Choose random salt r
  - Hash m and r together
  - hash-value = H_r(m)

- Useful for digital signatures
  - Signer chooses fresh salt for each signature
  - Protects against collision attacks
    - More on that later
What we propose

- A randomized mode-of-operation
  - Applicable to iterated hash functions
  - No changes to underlying hash functions

- Resists off-line collision attacks
  - Provably: only need something close to $2^{nd}$ pre-image resistance, not full collision-resistance [Crypto’06]
  - Attack is inherently on-line

- Use for signatures
  - No changes to sig algorithms (RSA, DSA)
Why randomized hashing?

- **Safety net** in case our hash functions are not as strong as we think
  - Much like HMAC does for MAC/PRF
  - Prudent engineering: adds another major line of defense against cryptanalysis

- Complements search for better hash functions, doesn’t replace it
Why now?

- Changes in standards, implementations are coming our way
- Even moving to SHA-2 takes significant effort (cf. [Bellovin-Rescorla])
- Residual effort to also support RMX is small in comparison
  - Small overhead, significant returns
New since last time

- Slightly modified the proposed mode
  \[ H_r(m_1 | \ldots | m_L) = H(r | m_1 \oplus r | \ldots | m_L \oplus r) \]
  - The new thing: \( r \) at the beginning

- **Signatures don’t need to “sign the salt”!**
  - Sufficient to sign only the hash value
    - Same as with deterministic hashing
  - Greatly simplifies implementations/deployment
    - No need to change encoding for signatures, etc.
The RMX transform

- **RMX**: message-randomization transform
  - \( \text{RMX}(r, m_1 | ... | m_L) = r | m_1 \oplus r | ... | m_L \oplus r \)
  - + rules for padding, etc.

- Can be used with any hash function
  - This is a mode-of-operation
  - E.g., RMX-SHA1, RMX-SHA256, etc.

- Should be standardized on its own
  - separately from individual hash functions
Analogous to CBC

- Mode-of-operation
- Can be used with any cipher
- Requires an additional input (the IV)
  - IV generation, transmission, etc. handled by the applications
  - Different applications handle the IV differently
Implementing RMX: test cases

- **Modified openssl**
  - Support for RMX in signatures
  - Use it for certificates

- **XML-signatures:**
  - RMX implemented by Michael McIntosh (IBM)
  - Can work with XML-sig’s “two-level hashing”

- See additional slides for details

- S/MIME, PGP, are next on our list

---

Less than 100 LOC due to the randomness, the rest would have to be done also for any new deterministic hashing
Feedback is Appreciated

- Feedback/suggestions regarding using RMX in other applications

Thank you for your attention
Additional Slides
Modifying `openssl`

This is needed also when adding a new deterministic hash function

- Hardest part: adding OIDs, changing config files to compile, link new functions
  - Changes in 10-15 files
- Implementing RMX: 2 new files (~360 LOC)
- Support for RMX signatures
  - ~40 LOC changed in `evp/evp.h`, `evp/digest.c`
- Use RMX for certificates
  - ~40 LOC changed in `asn1/a_sign.c`, `asn1/a_verify.c`

This is unique to RMX
Support for RMX signatures

- Signature interface in openssl:
  
  - EVP_SignInit, EVP_SignUpdate, EVP_SignFinal
  - EVP_VerifyInit, EVP_VerifyUpdate, EVP_VerifyFinal
  
  - Init/Update just macros for DigestInit/Update

- New Init interfaces
  
  - EVP_DigestInit_ex2(ctx, MD-type, engine, new-param)
  
  - Macros EVP_SignInit_ex2/VerifyInit_ex2

- New OIDs (types) for randomized hashing
Inserting RMX to control-flow

- Added “transform-needed” flag to MD-type (and param field to MD context)
- `DigestInit/Update/Final` check flag
  - If set, call `RMX_Init/Update/Final` rather than the underlying MD functions
  - `RMX_` functions do transform (using param), then call underlying MD functions
Using RMX for Certificates

- Signing/verifying from ASN1 modules
  
  ```c
  ASN1_item_verify(ASN1_ITEM *it, X509_ALGOR *a,
                   ASN1_BIT_STRING *signature, void *asn,
                   EVP_PKEY *pkey)
  
  - ASN1_item_sign is similar
  
  - The salt is passed inside X509_ALGOR
    
    - Parameter of the RMX-SHA1-RSA algorithm
  
  - ASN1_item_verify calls the new Init interface
    
    EVP_VerifyInitInitex2(..., salt)
XML Signatures

- Include transforms that are applied to data before hashing/signing
- Just add the RMX transform
  - Must be last transform before hashing
- Done by application, no change to signing code
  
  ```cpp
  // Do other transformations (envelope, canonicalize)
  RMX = get_a_pointer_to_implementation("URI-of-RMX");
  salt = call_your_favorite_RNG();
  x.addTransform(RMX, salt);
  // Proceed as usual
  ```
XML Signatures: 2-level Hashing

- XML sigs use a 2-level hashing scheme
  1. Each document is hashed to get digest
  2. Digests concatenated and hashed again
  3. Result is signed

- Part 2 does not have transforms
  - But it has canonicalization
  - Can write new canonicalization method that includes RMX
Aside: “first-party attacks”

- Can signer itself find collisions?
  - Only if hash is not collision-resistant
  - And even then non-repudiation is not effected
    - If signature is valid, signer is responsible
  - Most apps are not affected (e.g., certificates)

- Use RMX with a “strong hash function” $H$
  - If $H$ is strong then all is dandy
  - If $H$ is weaker than we initially thought, most applications are still protected