Strengthening Digital Signatures via Randomized Hashing

Shai Halevi and Hugo Krawczyk

IBM Research

Coping with Collisions

- Post-Wang Trauma (collision attacks):
  - A healthy reminder of our "shaky foundations"
- Applications threatened by collisions: mainly signatures
- What to do: avoid patches, build stronger hash funct's
  - But do we know how?
- Our approach: "Hope for the Best, Plan for the Worst"
- BUILD APPLICATIONS ON AS WEAK AS POSSIBLE ASSUMPTIONS ON THE UNDERLYING HASH FUNCTIONS

Our Contribution

We randomize the signature processing such that:

- Signatures remain secure even if off-line collision attacks against hash are successful
- Attacker needs to be able to mount a variant of the much harder "second-preimage attack" (on compr. f.)
- Off-line attacks useless: Per-signature attack (**on line!**)
  - Attack can start only when per-signature randomness revealed
- HASH FUNCTION AND SIGNING ALG UNCHANGED!!

Too Good To Be True?

- Simple message randomization scheme
- Provable reduction to "second preimage resistance"

- NIST: SP 800-106 !
  - Internet Draft is coming (see our website)

RMX: Message Randomization Scheme

- From SIGN(Hash(**M**)) to SIGN(Hash(**RMX(r,M)**))
- RMX(r,M) : $M=(m_1,m_2,\ldots,m_L)$, **r** → $(\mathbf{r}, m_1 \oplus \mathbf{r}, m_2 \oplus \mathbf{r}, \ldots, m_L \oplus \mathbf{r})$
- In signatures:

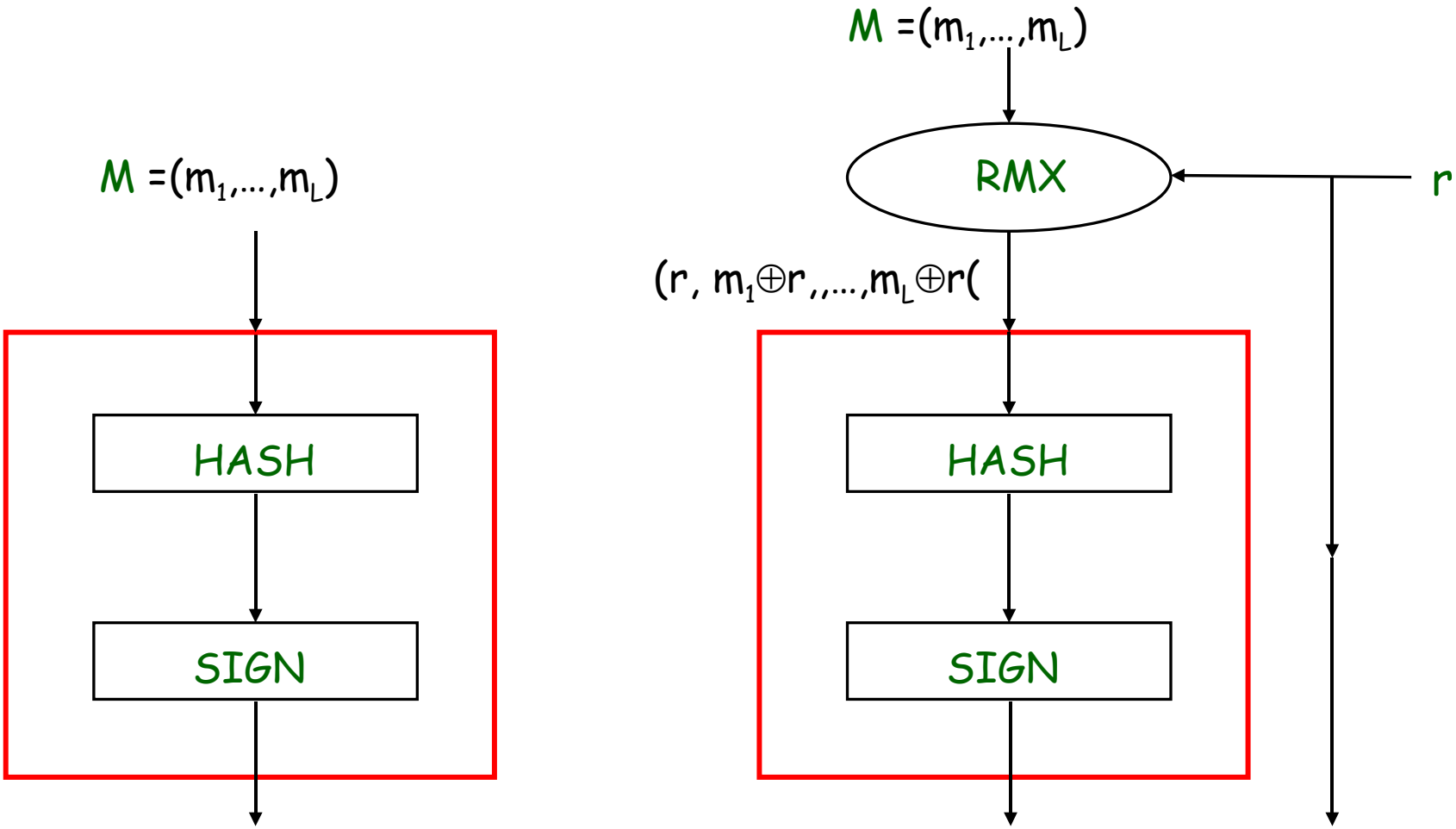
$M=(m_1,m_2,\ldots,m_L)$ … $m_2 \oplus \mathbf{r}, \ldots, m_L \oplus \mathbf{r})$ → SIGN, **r**

m_i and r of block length (eg 512)

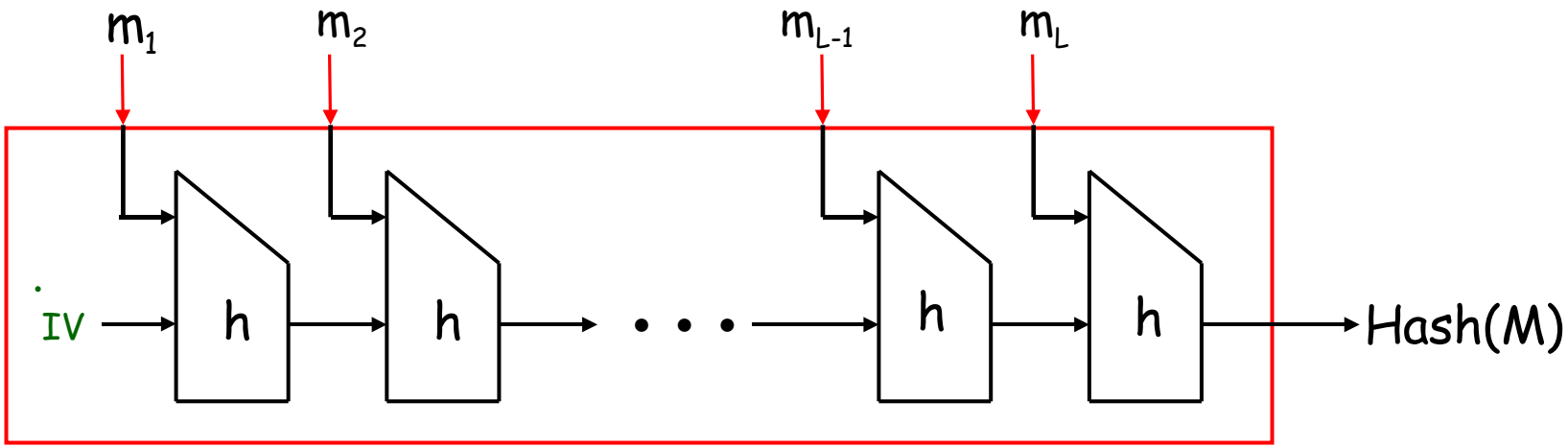
r chosen by signer at random w/each sig

Input to signing algorithm (r transmitted with sig)

RMX: Preserving Hash-then-Sign

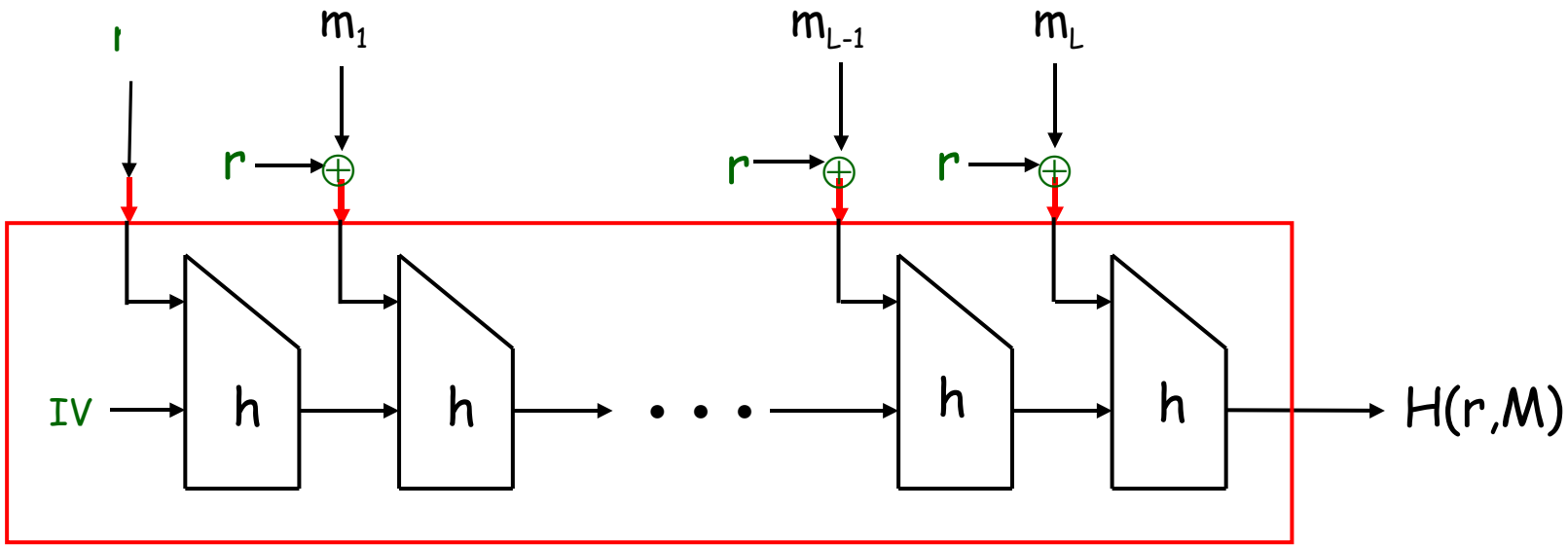


Merkle-Damgard Hash H



The RMX Scheme

(one-pass blockwise processing)



Practical

- RMX: simple front-end to existing hash-then-sign modules
- No change to hash functions or signature algorithms
- Compatible with block-wise processing of M-D functions
- Random generation by signer only (e.g., certificate issuing vs verifying)
  - 128 bits of randomness (up to a block, 512)
- Transporting r: application-dependent (like IV in CBC);
  - E.g., X.509: r as a parameter under AlgorithmIdentifier
- Implementations: certificate signing (openssl, NSS/Firefox[B&S])
  - XML: next (note: RMX can be applied to multilevel signing)
- Documented by NIST: SP 800-106 (Internet Draft coming)

Secure

- Substantial security increase for digital signatures
  - A fundamental shift in attack scenario: Off-line vs. On-line
    - In particular: no inherent birthday, shorter outputs (truncation)
  - A much harder cryptanalytical task (~SPR of compression function)
- Notes
  - Randomization never weakens: A SAFETY NET
  - Likely extension of useful life of hash functions, may prevent or mitigate catastrophic failure, more planning time upon weaknesses
  - Much like HMAC for MAC functions (btw, is HMAC good as RMX?)

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

- Paper (Crypto'06)
- Implementation experience
- Internet Draft

Note: Can the Signer Cheat?

- If H is CR then the signer cannot find collisions
- With RMX, if H is <u>not</u> CR, the signer (and only the signer!) may find r,r',M,M' s.t H(RMX(r,M))=H(RMX(r',M'))
- But this is no contradiction to non-repudiation
  - Signer is bound by any message with his signature (even if he shows two msgs with the same signature!)
  - NO contradiction to standard unforgeability definitions [GMR]
- Note: in RMX as long as H is CRHF then even the signer cannot find collisions (safety net!)

Note: Not *any* randomness…

- $H_r(M)=H(M||r)$: no help! needs collision resistance (same for r in the middle of msg)
- $H_r(M)=H(r||M)$: helps, but randomization fades on long msgs (contrast w/our scheme where r randomizes each block!)
- HMAC: $H(r||H(r||M))$ no better than previous

Randomized Hashing Implementation

- Java JCE Provider for java.security.Signature
  - ☹ No setParam for java.security.MessageDigest
- Apache XML Security library extensions
  - Signature
    - Salt parameter passed as child of SignatureMethod
  - Transform
    - Salt parameter passed as child of Transform

XML Signature Example


```
<Test><Data>Test Data</Data><ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo><ds:CanonicalizationMethod
        Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"></ds:CanonicalizationMethod>
      <ds:SignatureMethod Algorithm="http://www.research.ibm.com/rmx/xmldsig#rmx-rsa-sha1">
        <Salt xmlns="http://www.research.ibm.com/rmx/xmldsig">JYoVX6Pqdc/z/1k…</Salt></ds:SignatureMethod>
    <ds:Reference URI=""> <ds:Transforms>
      <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"></ds:Transform>
      <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"></ds:Transform>
        <ds:Transform Algorithm="http://www.research.ibm.com/rmx/xmldsig#rmx-sha1">
          <Salt xmlns="http://www.research.ibm.com/rmx/xmldsig">dS1mE6FG5IikizQEJKafg6kVChc=</Salt>
      </ds:Transform></ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
      <ds:DigestValue>xE/1oRdq+7z3KDAj9qh/GY/6SWQ=</ds:DigestValue>
    </ds:Reference></ds:SignedInfo>
  <ds:SignatureValue>fDDekndStUhk8wvfPJNe8pj0T2ZHPx4ZJ06s4kOhSvYucQCyNKUQrAdSQds…</ds:SignatureValue>
  <ds:KeyInfo><ds:KeyValue><ds:RSAKeyValue>
    <ds:Modulus>heazCYHwZC5kiGI6eO3ZSLjypfdgeXu3uXJoN/VYlWrP51NoJ5wR9NOnzAxChufT5qi0…</ds:Modulus>
    <ds:Exponent>AQAB</ds:Exponent></ds:RSAKeyValue></ds:KeyValue></ds:KeyInfo></ds:Signature>
</Test>
```

Adding Support for New SignatureMethod or DigestMethod

- Adding new JCE Signature Provider
  - Add one class derived from base; specify:
    - Underlying Signature Provider (e.g. DSA)
    - Associated MessageDigest block size (e.g. SHA1 = 20 bytes, MD5 = 16 bytes, etc.)
- Adding new Transform
  - Add one class derived from base; specify:
    - Underlying MessageDigest Provider