Digital Signatures

Digital Signatures

Lecture 13

Claude Shannon

Alan Turing

Merkle, Hellman & Diffie

Shamir , Rivest & Adleman
Turing Award ‘02

Manuel Blum
Turing Award ‘95

Andrew Yao
Turing Award ‘00

Goldwasser & Micali
Turing Award ‘12

Claude Shannon

Alan Turing

Merkle, Hellman & Diffie
Turing Award ‘15

Shamir , Rivest & Adleman
Turing Award ‘02

Manuel Blum
Turing Award ‘95

Andrew Yao
Turing Award ‘00

Goldwasser & Micali
Turing Award ‘12

Digital Signatures

Digital Signatures

- Syntax: KeyGen, $Sign_{SK}$ and $Verify_{VK}$.
  Security: Same experiment as MAC's, but adversary given VK

Digital Signatures

- Syntax: KeyGen, Sign_{SK} and Verify_{VK}.
  Security: Same experiment as MAC's, but adversary given VK
- Secure digital signatures using OWF, UOWHF and PRF

Digital Signatures

- Syntax: KeyGen, Sign_{SK} and Verify_{VK}.
  Security: Same experiment as MAC's, but adversary given VK
- Secure digital signatures using OWF, UOWHF and PRF
  - Hence, from OWF alone (more efficiently from OWP)

Digital Signatures

- Syntax: KeyGen, Sign_{SK} and Verify_{VK}.
  Security: Same experiment as MAC's, but adversary given VK
- Secure digital signatures using OWF, UOWHF and PRF
  - Hence, from OWF alone (more efficiently from OWP)
- More efficient using CRHF instead of UOWHF

Digital Signatures

- Syntax: KeyGen, $Sign_{SK}$ and $Verify_{VK}$.
  Security: Same experiment as MAC's, but adversary given VK
- Secure digital signatures using OWF, UOWHF and PRF
  - Hence, from OWF alone (more efficiently from OWP)
- More efficient using CRHF instead of UOWHF
- Even more efficient based on (strong) number-theoretic assumptions

Digital Signatures

- Syntax: KeyGen, Sign_{SK} and Verify_{VK}.
  Security: Same experiment as MAC's, but adversary given VK
- Secure digital signatures using OWF, UOWHF and PRF
  - Hence, from OWF alone (more efficiently from OWP)
- More efficient using CRHF instead of UOWHF
- Even more efficient based on (strong) number-theoretic assumptions
  - e.g. Cramer-Shoup Signature based on "Strong RSA assumption"

Digital Signatures

- Syntax: KeyGen, Sign_{SK} and Verify_{VK}.
  Security: Same experiment as MAC's, but adversary given VK
- Secure digital signatures using OWF, UOWHF and PRF
  - Hence, from OWF alone (more efficiently from OWP)
- More efficient using CRHF instead of UOWHF
- Even more efficient based on (strong) number-theoretic assumptions
  - e.g. Cramer-Shoup Signature based on "Strong RSA assumption"
- Efficient schemes secure in the Random Oracle Model

Digital Signatures

- Syntax: KeyGen, Sign_{SK} and Verify_{VK}.
  Security: Same experiment as MAC's, but adversary given VK
- Secure digital signatures using OWF, UOWHF and PRF
  - Hence, from OWF alone (more efficiently from OWP)
- More efficient using CRHF instead of UOWHF
- Even more efficient based on (strong) number-theoretic assumptions
  - e.g. Cramer-Shoup Signature based on "Strong RSA assumption"
- Efficient schemes secure in the Random Oracle Model
  - e.g. RSA-PSS in RSA Standard PKCS#1

One-time Digital Signatures

One-time Digital Signatures

- Recall One-time MAC to sign a single n bit message

One-time Digital Signatures

- Recall One-time MAC to sign a single n bit message
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$
  - Signature for $m_1 \ldots m_n$ be $(r^i_{m_i})_{i=1..n}$

r^1_0	r^2_0	r^3_0
r^1_1	r^2_1	r^3_1

One-time Digital Signatures

- Recall One-time MAC to sign a single n bit message
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$
  - Signature for $m_1...m_n$ be $(r^i_{m_i})_{i=1..n}$
- One-Time Digital Signature: Same signing key and signature, but VK= $(f(r^i_0),f(r^i_1))_{i=1..n}$ where f is a OWF

r^1_0	r^2_0	r^3_0
r^1_1	r^2_1	r^3_1

One-time Digital Signatures

- Recall One-time MAC to sign a single n bit message
  - Shared secret key: 2n random strings (each k-bit long) $(r^i{}_0, r^i{}_1)_{i=1..n}$
  - Signature for $m_1 \ldots m_n$ be $(r^i{}_{m_i})_{i=1..n}$
- One-Time Digital Signature: Same signing key and signature, but VK= $(f(r^i{}_0), f(r^i{}_1))_{i=1..n}$ where f is a OWF

$f(r^1{}_0)$	$f(r^2{}_0)$	$f(r^3{}_0)$
$f(r^1{}_1)$	$f(r^2{}_1)$	$f(r^3{}_1)$

$r^1{}_0$	$r^2{}_0$	$r^3{}_0$
$r^1{}_1$	$r^2{}_1$	$r^3{}_1$

One-time Digital Signatures

- Recall One-time MAC to sign a single n bit message
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$
  - Signature for $m_1 \ldots m_n$ be $(r^i_{m_i})_{i=1..n}$
- One-Time Digital Signature: Same signing key and signature, but VK= $(f(r^i_0), f(r^i_1))_{i=1..n}$ where f is a OWF
  - Verification applies f to signature elements and compares with VK

$f(r^1_0)$	$f(r^2_0)$	$f(r^3_0)$
$f(r^1_1)$	$f(r^2_1)$	$f(r^3_1)$

r^1_0	r^2_0	r^3_0
r^1_1	r^2_1	r^3_1

One-time Digital Signatures

- Recall One-time MAC to sign a single n bit message
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$
  - Signature for $m_1 \ldots m_n$ be $(r^i_{m_i})_{i=1..n}$
- One-Time Digital Signature: Same signing key and signature, but VK= $(f(r^i_0), f(r^i_1))_{i=1..n}$ where f is a OWF
  - Verification applies f to signature elements and compares with VK
  - Security [Exercise]

$f(r^1_0)$	$f(r^2_0)$	$f(r^3_0)$
$f(r^1_1)$	$f(r^2_1)$	$f(r^3_1)$

r^1_0	r^2_0	r^3_0
r^1_1	r^2_1	r^3_1

One-time Digital Signatures

Lamport's One-Time Signature

- Recall One-time MAC to sign a single n bit message
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$
  - Signature for $m_1 \ldots m_n$ be $(r^i_{m_i})_{i=1..n}$
- One-Time Digital Signature: Same signing key and signature, but VK= $(f(r^i_0), f(r^i_1))_{i=1..n}$ where f is a OWF
  - Verification applies f to signature elements and compares with VK
  - Security [Exercise]

$f(r^1_0)$	$f(r^2_0)$	$f(r^3_0)$
$f(r^1_1)$	$f(r^2_1)$	$f(r^3_1)$

r^1_0	r^2_0	r^3_0
r^1_1	r^2_1	r^3_1

Domain Extension of (One-time) Signatures

Domain Extension of (One-time) Signatures

- Lamport's scheme has a fixed-length message (and SK/VK are much longer than the message)

Domain Extension of (One-time) Signatures

- Lamport's scheme has a fixed-length message (and SK/VK are much longer than the message)
- Hash-and-Sign domain extension

Domain Extension of (One-time) Signatures

- Lamport's scheme has a fixed-length message (and SK/VK are much longer than the message)
- Hash-and-Sign domain extension
  - (If applied to one-time signature, still one-time, but with variable input-length)

Domain Extension of (One-time) Signatures

- Lamport's scheme has a fixed-length message (and SK/VK are much longer than the message)
- Hash-and-Sign domain extension
  - (If applied to one-time signature, still one-time, but with variable input-length)
  - Domain extension using a CRHF (not weak CRHF, unlike for MAC)

Domain Extension of (One-time) Signatures

- Lamport's scheme has a fixed-length message (and SK/VK are much longer than the message)
- Hash-and-Sign domain extension
  - (If applied to one-time signature, still one-time, but with variable input-length)
  - Domain extension using a CRHF (not weak CRHF, unlike for MAC)
    - $\text{Sign}^*_{SK,h}(M) = \text{Sign}_{SK}(h(M))$ where $h \leftarrow \mathcal{H}$ in both SK*,VK*

Domain Extension of (One-time) Signatures

- Lamport's scheme has a fixed-length message (and SK/VK are much longer than the message)
- Hash-and-Sign domain extension
  - (If applied to one-time signature, still one-time, but with variable input-length)
  - Domain extension using a CRHF (not weak CRHF, unlike for MAC)
    - $\text{Sign}^*_{SK,h}(M) = \text{Sign}_{SK}(h(M))$ where $h \leftarrow \mathcal{H}$ in both SK*,VK*
  - Can use UOWHF, with fresh h every time (included in signature)

Domain Extension of (One-time) Signatures

- Lamport's scheme has a fixed-length message (and SK/VK are much longer than the message)
- Hash-and-Sign domain extension
  - (If applied to one-time signature, still one-time, but with variable input-length)
  - Domain extension using a CRHF (not weak CRHF, unlike for MAC)
    - $\text{Sign}^*_{SK,h}(M) = \text{Sign}_{SK}(h(M))$ where $h \leftarrow \mathcal{H}$ in both SK*,VK*
  - Can use UOWHF, with fresh h every time (included in signature)
    - $\text{Sign}^*_{SK}(M) = (\, h, \text{Sign}_{SK}(h,h(M))\,)$ where $h \leftarrow \mathcal{H}$ picked by signer

Domain Extension of (One-time) Signatures

- Lamport's scheme has a fixed-length message (and SK/VK are much longer than the message)
- Hash-and-Sign domain extension
  - (If applied to one-time signature, still one-time, but with variable input-length)
  - Domain extension using a CRHF (not weak CRHF, unlike for MAC)
    - $\text{Sign}^*_{SK,h}(M) = \text{Sign}_{SK}(h(M))$ where $h \leftarrow \mathcal{H}$ in both SK*,VK*
  - Can use UOWHF, with fresh h every time (included in signature)
    - $\text{Sign}^*_{SK}(M) = (\, h, \text{Sign}_{SK}(h, h(M)) \,)$ where $h \leftarrow \mathcal{H}$ picked by signer
    - This can then be used to build a full-fledged signature scheme starting from one-time signatures (skipped)

More Efficient Signatures

More Efficient Signatures

- Diffie-Hellman suggestion (heuristic): Sign(M) = $f^{-1}(M)$ where (SK,VK) = (f^{-1}, f), a Trapdoor OWP pair. Verify(M,σ) = 1 iff $f(\sigma)$=M.

More Efficient Signatures

- Diffie-Hellman suggestion (heuristic): Sign(M) = f^{-1}(M) where (SK,VK) = (f^{-1},f), a Trapdoor OWP pair. Verify(M,σ) = 1 iff f(σ)=M.
  - Attack: pick σ, let M=f(σ) (Existential forgery)

More Efficient Signatures

- Diffie-Hellman suggestion (heuristic): Sign(M) = $f^{-1}(M)$ where (SK,VK) = (f^{-1},f), a Trapdoor OWP pair. Verify(M,σ) = 1 iff f(σ)=M.
  - Attack: pick σ, let M=f(σ) (Existential forgery)
- Fix: **Sign(M) = f^{-1}(Hash(M))**

More Efficient Signatures

- Diffie-Hellman suggestion (heuristic): Sign(M) = $f^{-1}(M)$ where (SK,VK) = (f^{-1},f), a Trapdoor OWP pair. Verify(M,σ) = 1 iff $f(\sigma)=M$.
  - Attack: pick σ, let $M=f(\sigma)$ (Existential forgery)
- Fix: **Sign(M) = f^{-1}(Hash(M))**
  - Secure? Adversary gets to choose M and hence Hash(M), and so the signing oracle gives adversary access to f^{-1} oracle. But T-OWP gives no guarantees when adversary is given f^{-1} oracle.

More Efficient Signatures

- Diffie-Hellman suggestion (heuristic): Sign(M) = f^{-1}(M) where (SK,VK) = (f^{-1},f), a Trapdoor OWP pair. Verify(M,σ) = 1 iff f(σ)=M.
  - Attack: pick σ, let M=f(σ) (Existential forgery)
- Fix: **Sign(M) = f^{-1}(Hash(M))**
  - Secure? Adversary gets to choose M and hence Hash(M), and so the signing oracle gives adversary access to f^{-1} oracle. But T-OWP gives no guarantees when adversary is given f^{-1} oracle.
  - If Hash(.) modeled as a random oracle then adversary can't choose Hash(M), and effectively doesn't have access to f^{-1} oracle. Then indeed secure

More Efficient Signatures

- Diffie-Hellman suggestion (heuristic): Sign(M) = f^{-1}(M) where (SK,VK) = (f^{-1},f), a Trapdoor OWP pair. Verify(M,σ) = 1 iff f(σ)=M.
  - Attack: pick σ, let M=f(σ) (Existential forgery)
- Fix: **Sign(M) = f^{-1}(Hash(M))**
  - Secure? Adversary gets to choose M and hence Hash(M), and so the signing oracle gives adversary access to f^{-1} oracle. But T-OWP gives no guarantees when adversary is given f^{-1} oracle.
  - If Hash(.) modeled as a random oracle then adversary can't choose Hash(M), and effectively doesn't have access to f^{-1} oracle. Then indeed secure
  - "Standard schemes" like RSA-PSS are based on this

Proving Security in the RO Model

Proving Security in the RO Model

- To prove that if T-OWP secure, then Sign(M) = f^{-1}(Hash(M)) is a secure digital signature in the RO Model, if Hash is a random oracle

Proving Security in the RO Model

- To prove that if T-OWP secure, then Sign(M) = f^{-1}(Hash(M)) is a secure digital signature in the RO Model, if Hash is a random oracle
  - Intuition: adversary only sees (x,f^{-1}(x)) where x is random, which it could have obtained anyway, by picking f^{-1}(x) first

Proving Security in the RO Model

- To prove that if T-OWP secure, then Sign(M) = f^{-1}(Hash(M)) is a secure digital signature in the RO Model, if Hash is a random oracle
    - Intuition: adversary only sees $(x,f^{-1}(x))$ where x is random, which it could have obtained anyway, by picking $f^{-1}(x)$ first
- Modeling as an RO: RO randomly initialized to a random function H from $\{0,1\}^*$ to $\{0,1\}^k$

Proving Security in the RO Model

- To prove that if T-OWP secure, then Sign(M) = f^{-1}(Hash(M)) is a secure digital signature in the RO Model, if Hash is a random oracle
  - Intuition: adversary only sees $(x,f^{-1}(x))$ where x is random, which it could have obtained anyway, by picking $f^{-1}(x)$ first

H an infinite object

- Modeling as an RO: RO randomly initialized to a random function H from $\{0,1\}^*$ to $\{0,1\}^k$

Proving Security in the RO Model

- To prove that if T-OWP secure, then Sign(M) = f^{-1}(Hash(M)) is a secure digital signature in the RO Model, if Hash is a random oracle
    - Intuition: adversary only sees (x,f^{-1}(x)) where x is random, which it could have obtained anyway, by picking f^{-1}(x) first

H an infinite object

- Modeling as an RO: RO randomly initialized to a random function H from $\{0,1\}^*$ to $\{0,1\}^k$
    - Signer and verifier (and forger) get oracle access to H(.)

Proving Security in the RO Model

- To prove that if T-OWP secure, then Sign(M) = f^{-1}(Hash(M)) is a secure digital signature in the RO Model, if Hash is a random oracle
  - Intuition: adversary only sees (x,f^{-1}(x)) where x is random, which it could have obtained anyway, by picking f^{-1}(x) first

H an infinite object

- Modeling as an RO: RO randomly initialized to a random function H from $\{0,1\}^*$ to $\{0,1\}^k$
  - Signer and verifier (and forger) get oracle access to H(.)
  - All probabilities also over the initialization of the RO

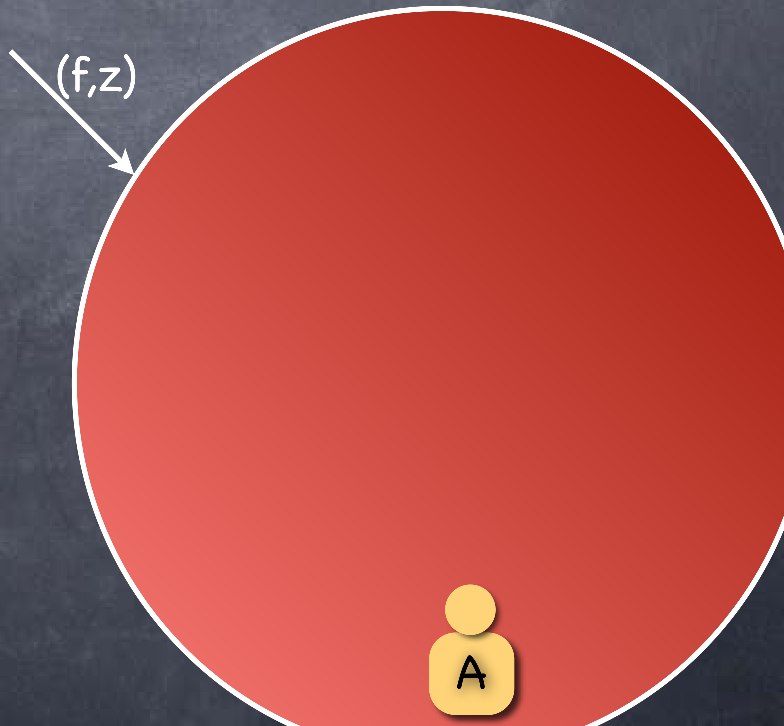
Proving Security in ROM

Proving Security in ROM

- Reduction: If A forges signature (where Sign(M) = $f^{-1}(H(M))$ with (f,f^{-1}) from TOWP and H an RO), then A* that can break T-OWP (i.e., given just f, and a random challenge z, can find $f^{-1}(z)$ w.n.n.p). A*(f,z) runs A internally.

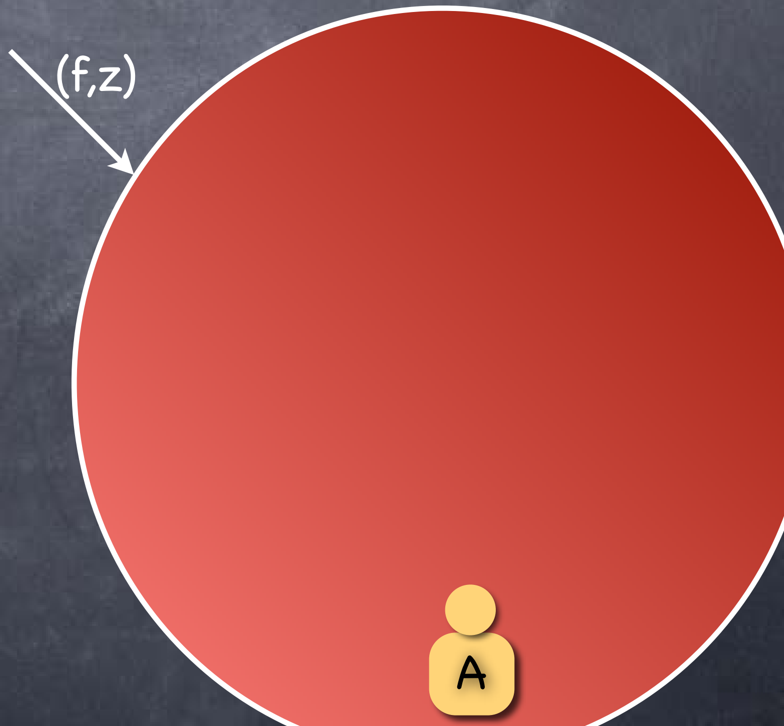
Proving Security in ROM

- Reduction: If A forges signature (where Sign(M) = $f^{-1}(H(M))$ with (f,f^{-1}) from TOWP and H an RO), then A* that can break T-OWP (i.e., given just f, and a random challenge z, can find $f^{-1}(z)$ w.n.n.p). A*(f,z) runs A internally.



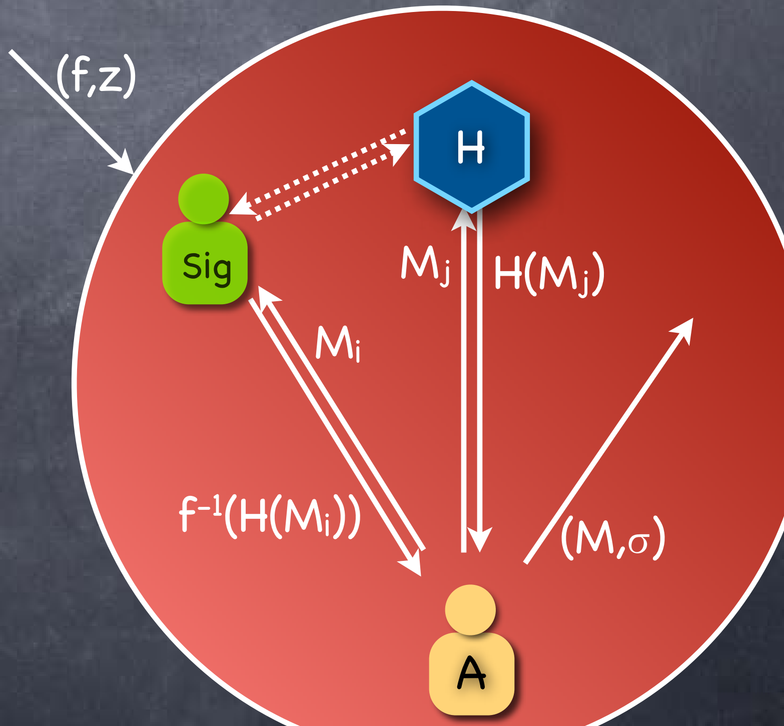
Proving Security in ROM

- Reduction: If A forges signature (where Sign(M) = $f^{-1}(H(M))$ with (f,f^{-1}) from TOWP and H an RO), then A* that can break T-OWP (i.e., given just f, and a random challenge z, can find $f^{-1}(z)$ w.n.n.p). A*(f,z) runs A internally.
  - A expects f, access to the RO and a signing oracle f^{-1}(Hash(.)) and outputs (M,σ) as forgery



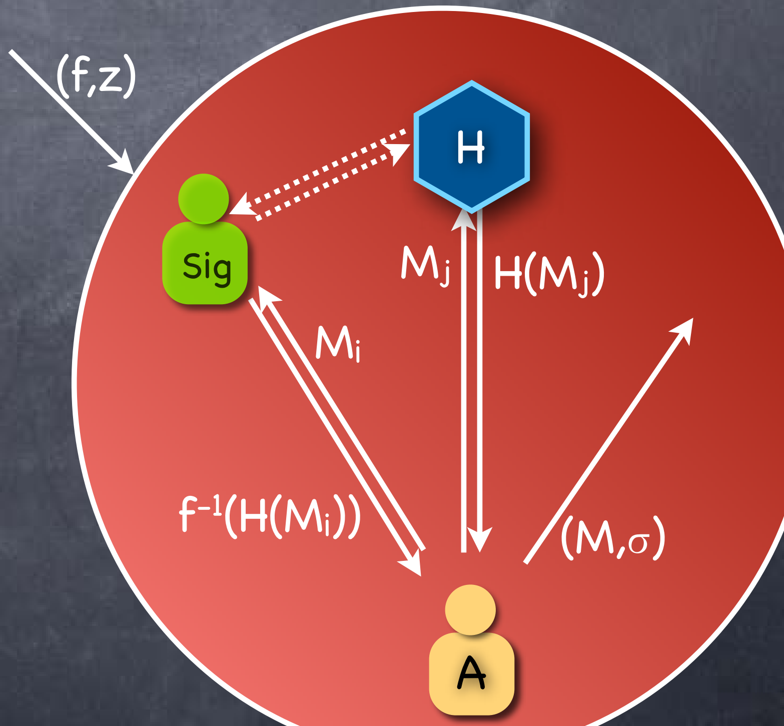
Proving Security in ROM

- Reduction: If A forges signature (where Sign(M) = $f^{-1}(H(M))$ with (f,f^{-1}) from TOWP and H an RO), then A* that can break T-OWP (i.e., given just f, and a random challenge z, can find $f^{-1}(z)$ w.n.n.p). A*(f,z) runs A internally.
  - A expects f, access to the RO and a signing oracle f^{-1}(Hash(.)) and outputs (M,σ) as forgery



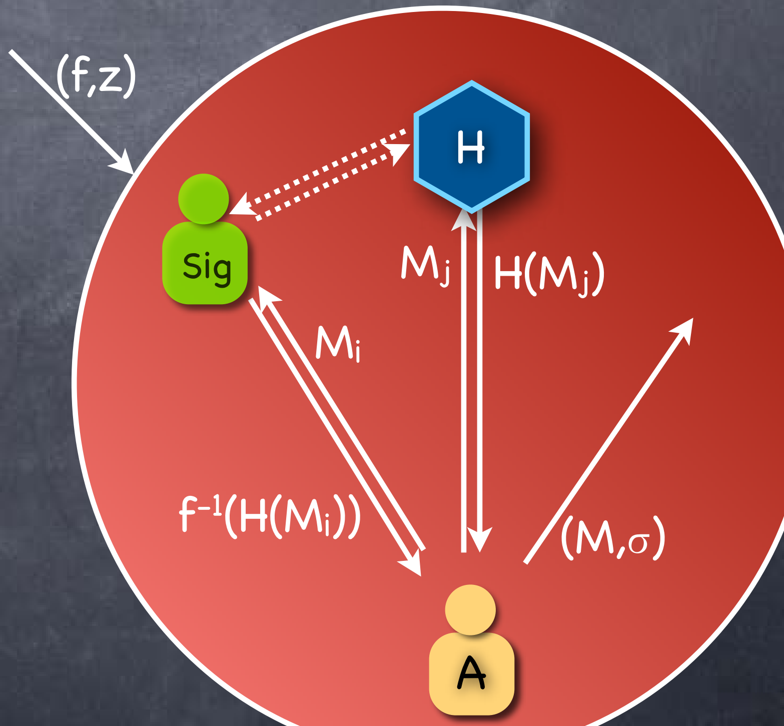
Proving Security in ROM

- Reduction: If A forges signature (where Sign(M) = $f^{-1}(H(M))$ with (f,f^{-1}) from TOWP and H an RO), then A* that can break T-OWP (i.e., given just f, and a random challenge z, can find $f^{-1}(z)$ w.n.n.p). A*(f,z) runs A internally.
  - A expects f, access to the RO and a signing oracle f^{-1}(Hash(.)) and outputs (M,σ) as forgery
  - A* can implement RO: a random response to each new query!



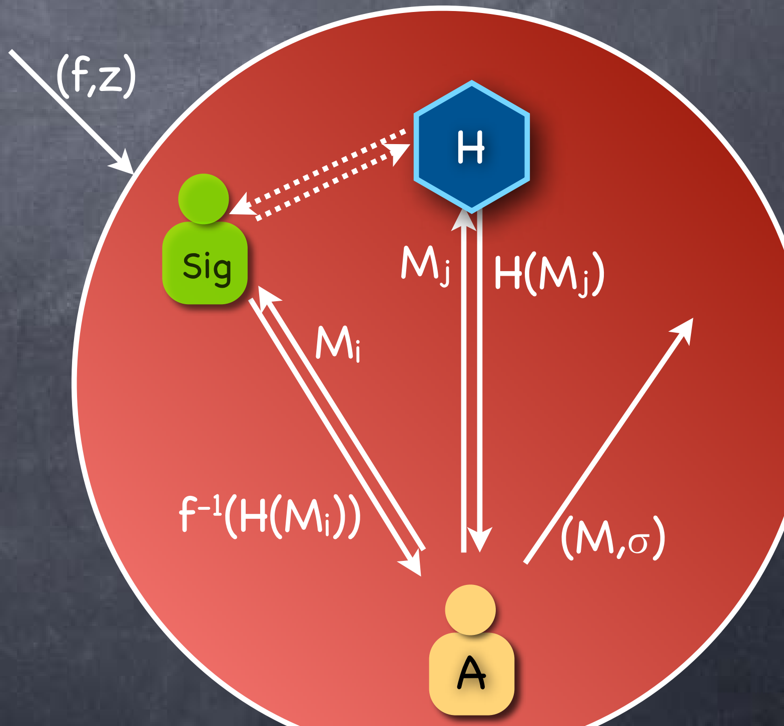
Proving Security in ROM

- Reduction: If A forges signature (where Sign(M) = $f^{-1}(H(M))$ with (f,f^{-1}) from TOWP and H an RO), then A* that can break T-OWP (i.e., given just f, and a random challenge z, can find $f^{-1}(z)$ w.n.n.p). A*(f,z) runs A internally.
  - A expects f, access to the RO and a signing oracle $f^{-1}(Hash(.))$ and outputs (M,σ) as forgery
  - A* can implement RO: a random response to each new query!
  - A* gets f, but doesn't have f^{-1} to sign



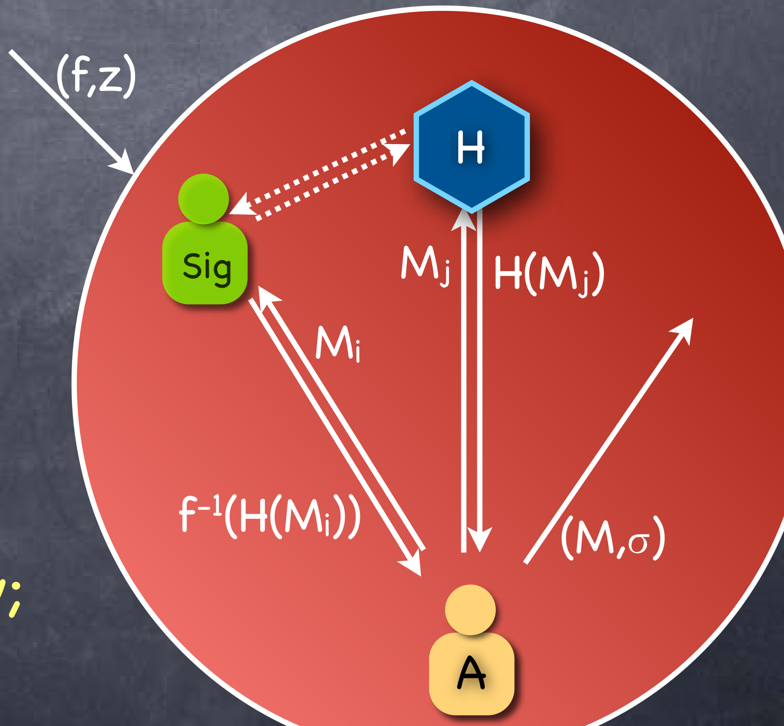
Proving Security in ROM

- Reduction: If A forges signature (where Sign(M) = $f^{-1}(H(M))$ with (f,f^{-1}) from TOWP and H an RO), then A* that can break T-OWP (i.e., given just f, and a random challenge z, can find $f^{-1}(z)$ w.n.n.p). A*(f,z) runs A internally.
  - A expects f, access to the RO and a signing oracle $f^{-1}(Hash(.))$ and outputs (M,σ) as forgery
  - A* can implement RO: a random response to each new query!
  - A* gets f, but doesn't have f^{-1} to sign
    - But x = H(M) is a random value that A* can pick!

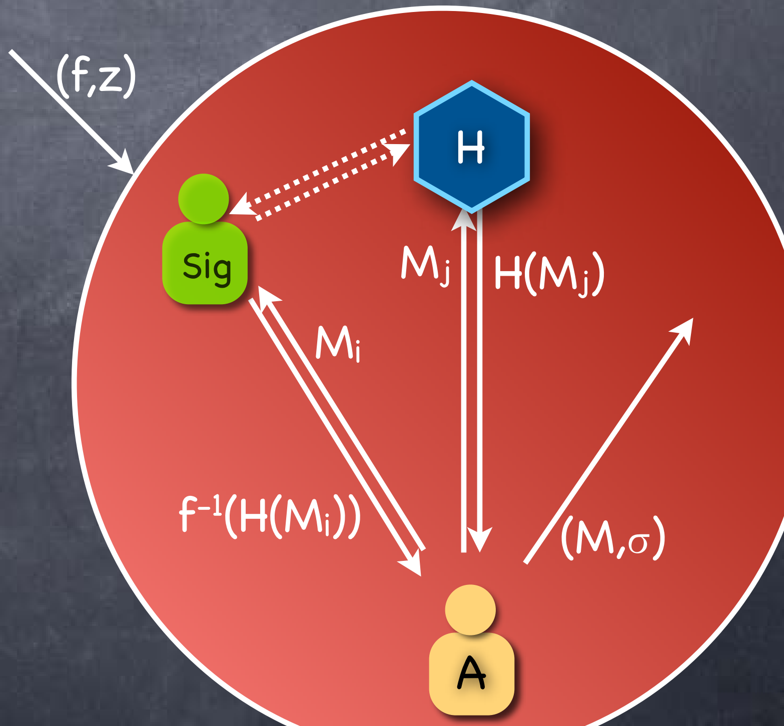


Proving Security in ROM

- Reduction: If A forges signature (where Sign(M) = $f^{-1}(H(M))$ with (f,f^{-1}) from TOWP and H an RO), then A* that can break T-OWP (i.e., given just f, and a random challenge z, can find $f^{-1}(z)$ w.n.n.p). A*(f,z) runs A internally.
  - A expects f, access to the RO and a signing oracle f^{-1}(Hash(.)) and outputs (M,σ) as forgery
  - A* can implement RO: a random response to each new query!
  - A* gets f, but doesn't have f^{-1} to sign
    - But x = H(M) is a random value that A* can pick!
    - A* picks H(M) as x=f(y) for random y; then Sign(M) = $f^{-1}(x)$ = y

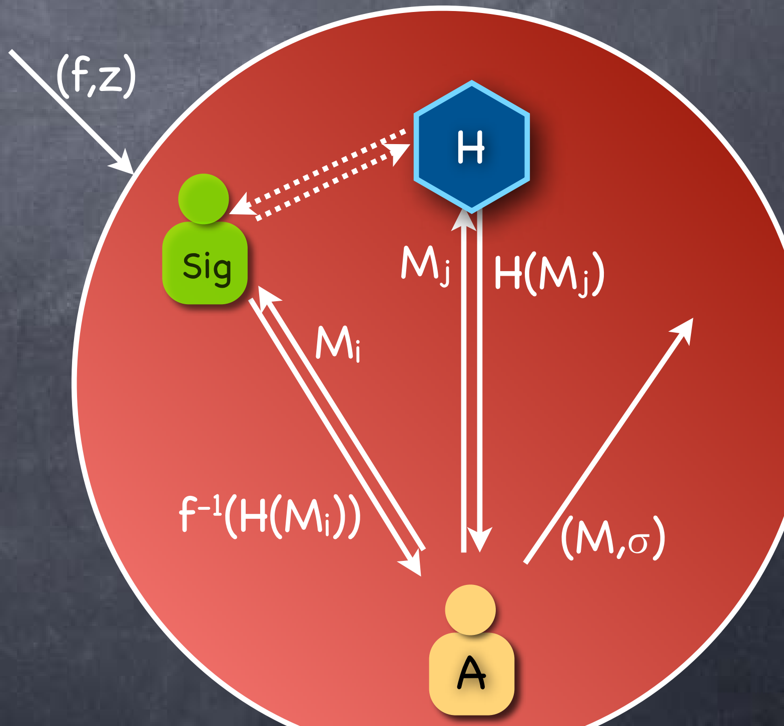


Proving Security in ROM



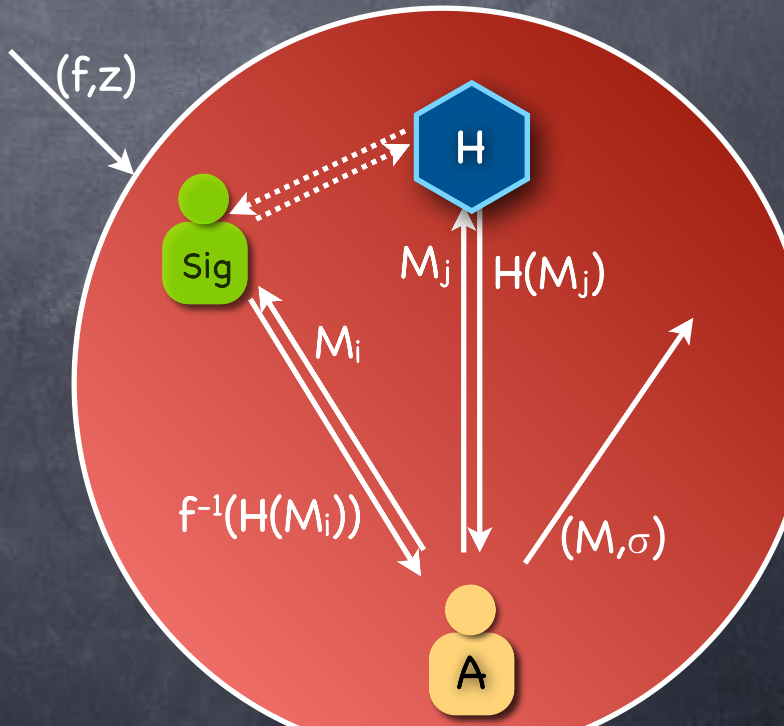
Proving Security in ROM

- A* such that if A forges signature, then A* can break T-OWP



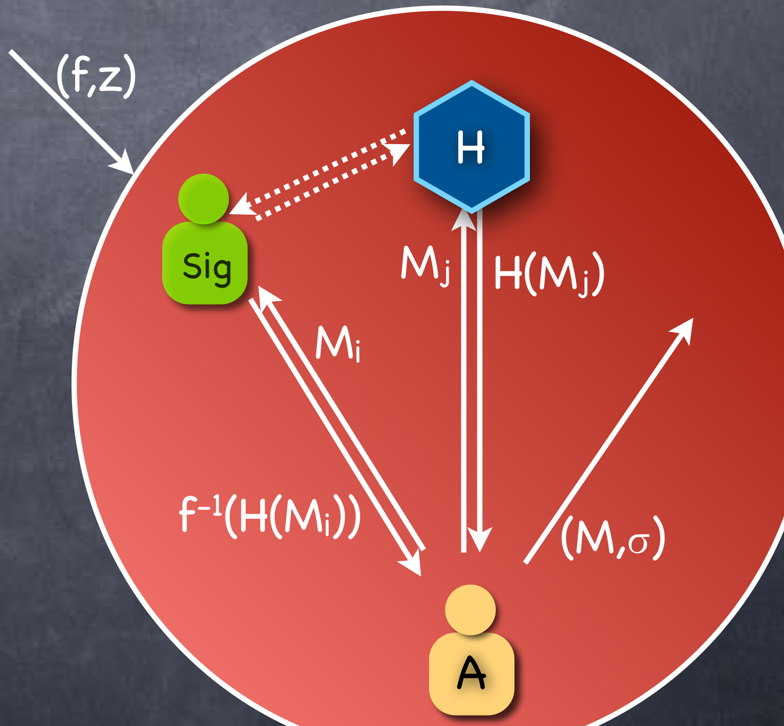
Proving Security in ROM

- A* such that if A forges signature, then A* can break T-OWP
  - A* implements H and Sign: For each new M queried to H or Sign, A* sets H(M)=f(y) for random y; then Sign(M) = y



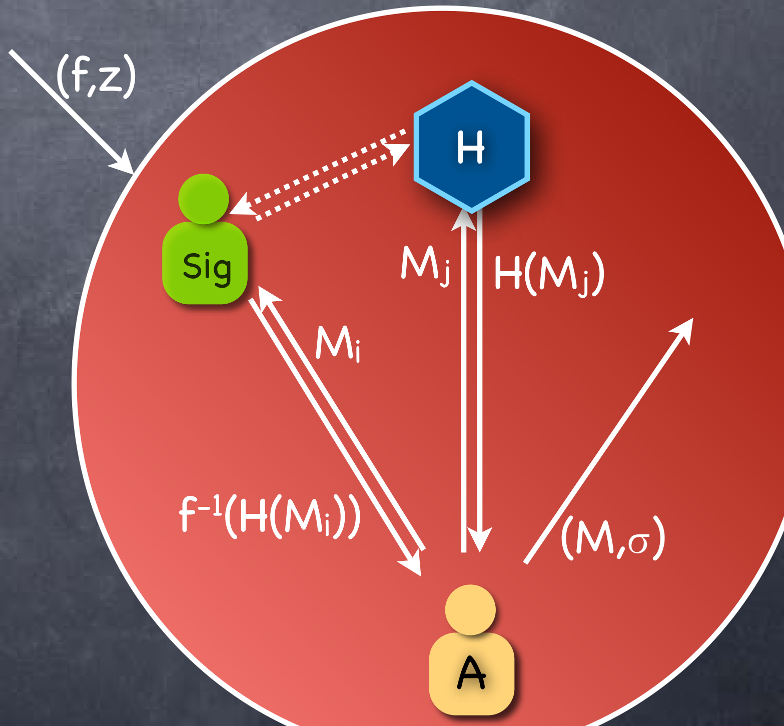
Proving Security in ROM

- A^* such that if A forges signature, then A^* can break T-OWP
  - A^* implements H and Sign: For each new M queried to H or Sign, A^* sets H(M)=f(y) for random y; then Sign(M) = y
  - But A^* should force A to invert z



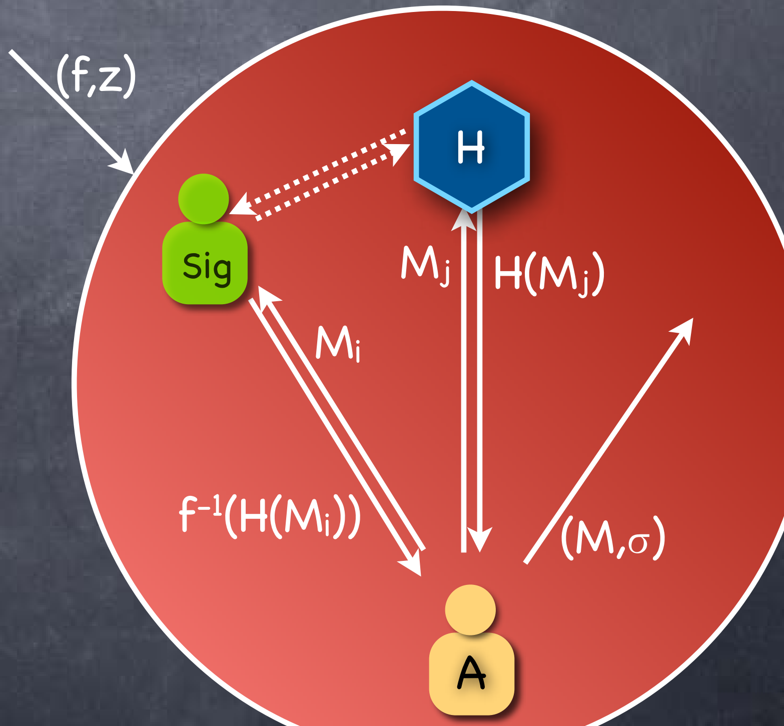
Proving Security in ROM

- A* such that if A forges signature, then A* can break T-OWP
  - A* implements H and Sign: For each new M queried to H or Sign, A* sets H(M)=f(y) for random y; then Sign(M) = y
  - But A* should force A to invert z
    - For a random (new) query M (say j^{th}) A* sets H(M)=z



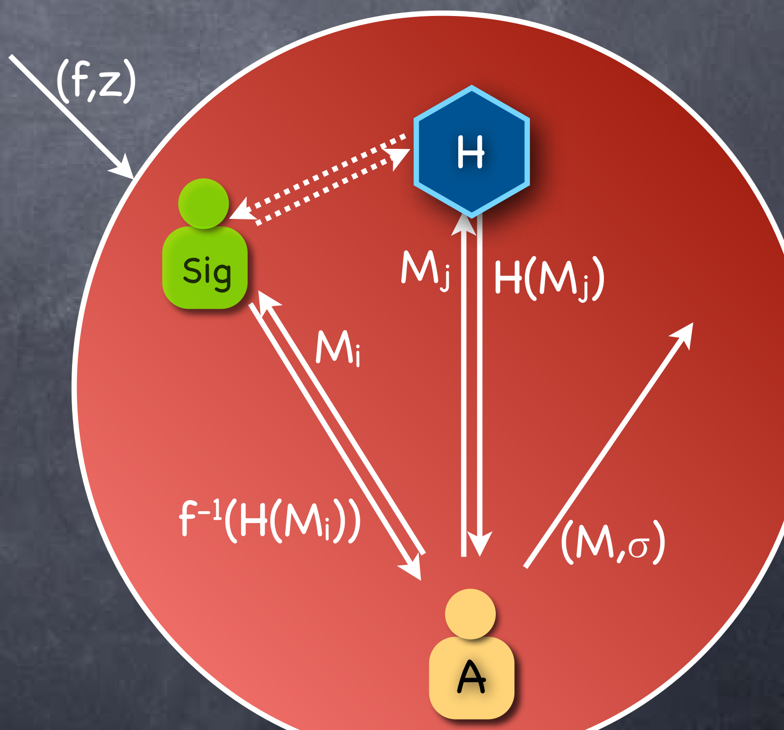
Proving Security in ROM

- A* such that if A forges signature, then A* can break T-OWP
  - A* implements H and Sign: For each new M queried to H or Sign, A* sets H(M)=f(y) for random y; then Sign(M) = y
  - But A* should force A to invert z
    - For a random (new) query M (say j^{th}) A* sets H(M)=z
      - Here queries include the "last query" to H, i.e., the one for verifying the forgery (may or may not be a new query)



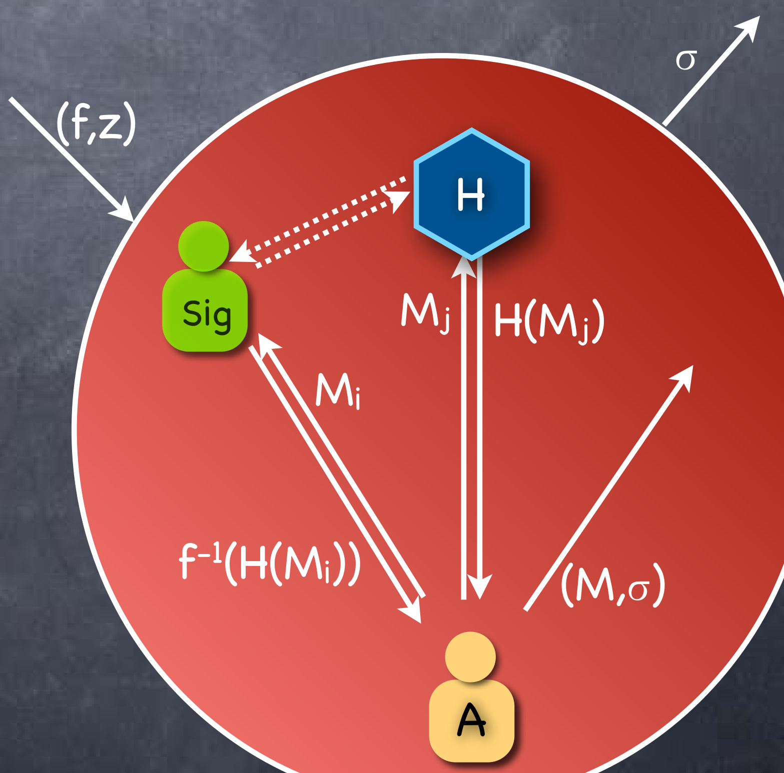
Proving Security in ROM

- A* such that if A forges signature, then A* can break T-OWP
  - A* implements H and Sign: For each new M queried to H or Sign, A* sets H(M)=f(y) for random y; then Sign(M) = y
  - But A* should force A to invert z
    - For a random (new) query M (say j^th) A* sets H(M)=z
      - Here queries include the "last query" to H, i.e., the one for verifying the forgery (may or may not be a new query)
- If q a bound on the number of queries that A makes to Sign/H, then with probability at least 1/q, A* would have set H(M)=z, where M is the message in the forgery



Proving Security in ROM

- A^* such that if A forges signature, then A^* can break T-OWP
  - A^* implements H and Sign: For each new M queried to H or Sign, A^* sets H(M)=f(y) for random y; then Sign(M) = y
  - But A^* should force A to invert z
    - For a random (new) query M (say j^{th}) A^* sets H(M)=z
      - Here queries include the "last query" to H, i.e., the one for verifying the forgery (may or may not be a new query)
- If q a bound on the number of queries that A makes to Sign/H, then with probability at least 1/q, A^* would have set H(M)=z, where M is the message in the forgery
  - In that case forgery $\Rightarrow \sigma = f^{-1}(z)$



Randomness Extraction

Randomness Extractor

Randomness Extractor

- Consider a PRG which outputs a pseudorandom group element in some complicated group

Randomness Extractor

- 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

Randomness Extractor

- 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...

Randomness Extractor

- 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

Randomness Extractor

- 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...

Randomness Extractor

- 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 uniform randomness? Depends on the specific dependencies...
- A general tool for purifying randomness: Randomness Extractor

Randomness Extractors

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
  - "Optimal" in all parameters except seed length

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
  - "Optimal" in all parameters except seed length
- Constructions with short seeds

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
  - "Optimal" in all parameters except seed length
- Constructions with short seeds
  - e.g. Based on expander graphs

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient <u>entropy</u>)
- 2-Universal Hash Functions
  - "Optimal" in all parameters except seed length
- Constructions with short seeds
  - e.g. Based on expander graphs
- Pseudorandomness Extractors: output is guaranteed only to be pseudorandom if input has sufficient (pseudo)entropy

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
  - "Optimal" in all parameters except seed length
- Constructions with short seeds
  - e.g. Based on expander graphs
- Pseudorandomness Extractors: output is guaranteed only to be pseudorandom if input has sufficient (pseudo)entropy
  - Can be based on iterated-hash functions or CBC-MAC

Randomness Extractors

- Statistical guarantees (output not just pseudorandom, but truly random, if input has sufficient entropy)
- 2-Universal Hash Functions
  - "Optimal" in all parameters except seed length
- Constructions with short seeds
  - e.g. Based on expander graphs
- Pseudorandomness Extractors: output is guaranteed only to be pseudorandom if input has sufficient (pseudo)entropy
  - Can be based on iterated-hash functions or CBC-MAC
  - Statistical guarantee, if compression function/block-cipher is a random function/random permutation (not random oracle)

Randomness Extractors

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed
  - 2-UHF is an example

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed
  - 2-UHF is an example
- Useful in key agreement

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed
  - 2-UHF is an example
- Useful in key agreement
  - Alice and Bob exchange a non-uniform key, with a lot of pseudoentropy for Eve (say, g^{xy})

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed
    - 2-UHF is an example
- 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

Randomness Extractors

- Strong extractor: output is random even when the seed for extraction is revealed
  - 2-UHF is an example
- 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)