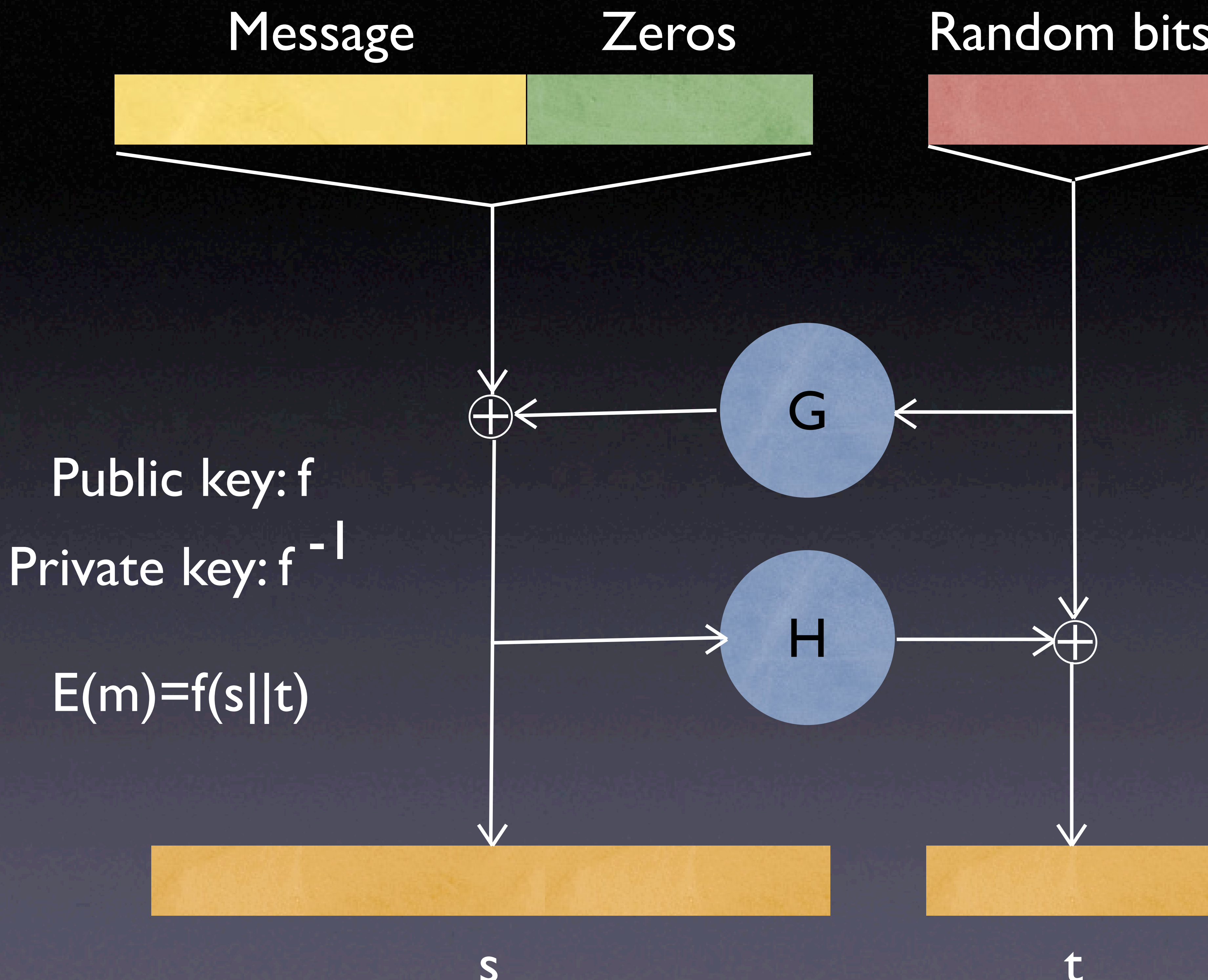
Your Suggestions

Given one random oracle $R(x)$ how can we make two random oracles $H(x)$ and $G(x)$

$$H(x) = R(0 \parallel x)$$
$$G(x) = R(1 \parallel x)$$

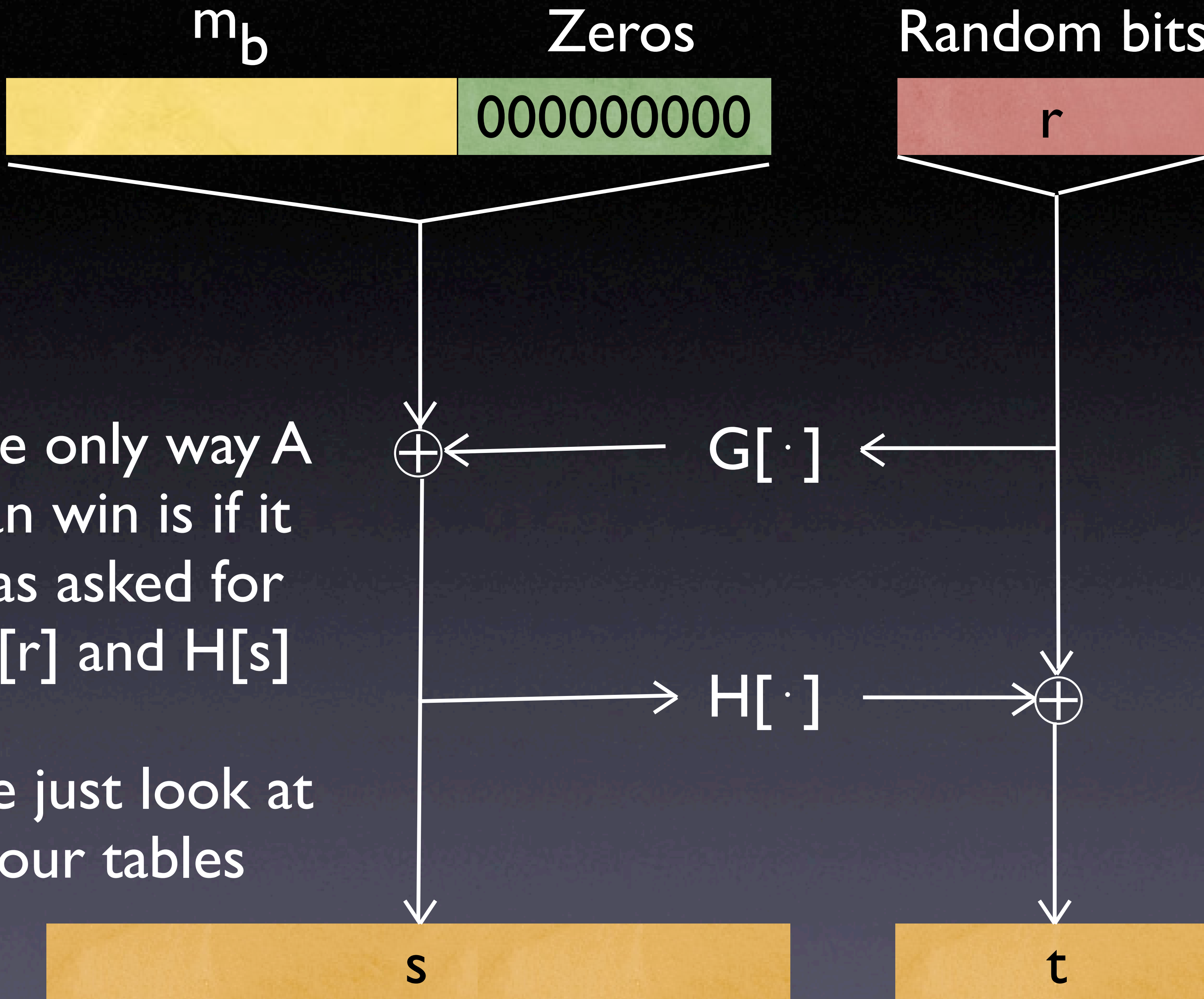

Message
Zeros
Random bits
G
H
Public key: f
Private key: f -1
E(m)=f(s||t)
s
t

Security of OAEP

- Construction and proof published in *Eurocrypt '94*
- Included in standards like SET (payment system proposed by Visa and Mastercard)

Early Objections

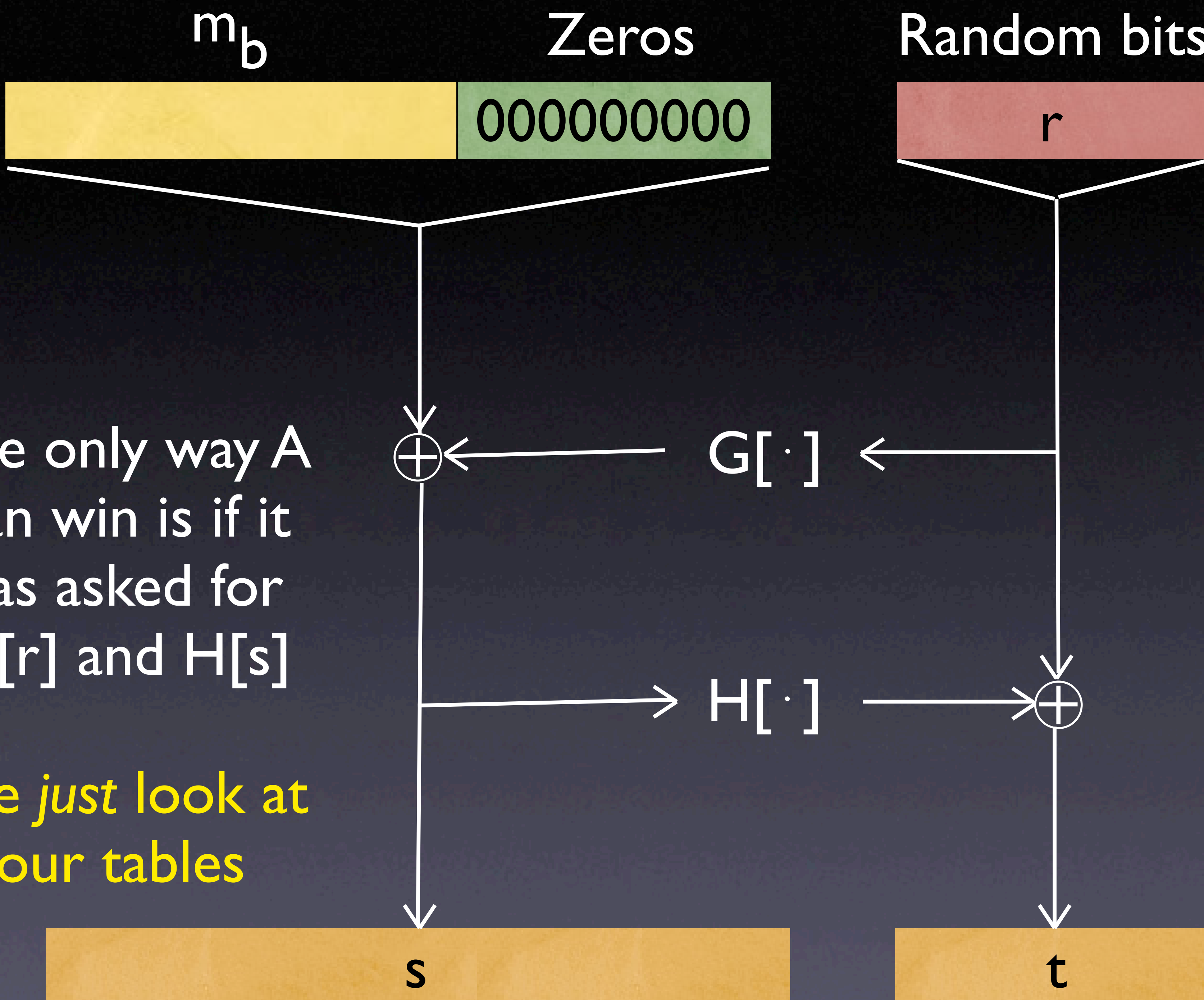
- Use random oracles, not really proving anything
- *Security bound* not tight enough
  - Proof says that if someone can break OAEP, can invert trapdoor permutation
  - Also tells how long it will take



The only way A can win is if it has asked for G[r] and H[s]

We just look at our tables

$$y = f(x) = f(s \,||\, t)$$



The only way A can win is if it has asked for G[r] and H[s]

We *just* look at our tables

$y = f(x) = f(s \,||\, t)$

Cost of Attack

- If adversary that breaks OAEP takes *n* steps:
  - He could ask for *n* different encryptions
  - Each encryption uses 2 oracle queries, i.e. one entry in each table
  - Trying all the combinations to break OWTP takes $O(n^2)$ operations

Cost of Attack

- If we want the attack on OAEP to take 2^{80} steps, we need the attack on the OWTP (e.g. RSA) to take at least 2^{160} steps
- With our best current attack on RSA, we'd need to use really big and inefficient keys (~5000 bit keys)

But...

- The proof is wrong
  - There's a hole in the argument
- There is a counter example
  - What we were trying to prove isn't even true

Proof of Security

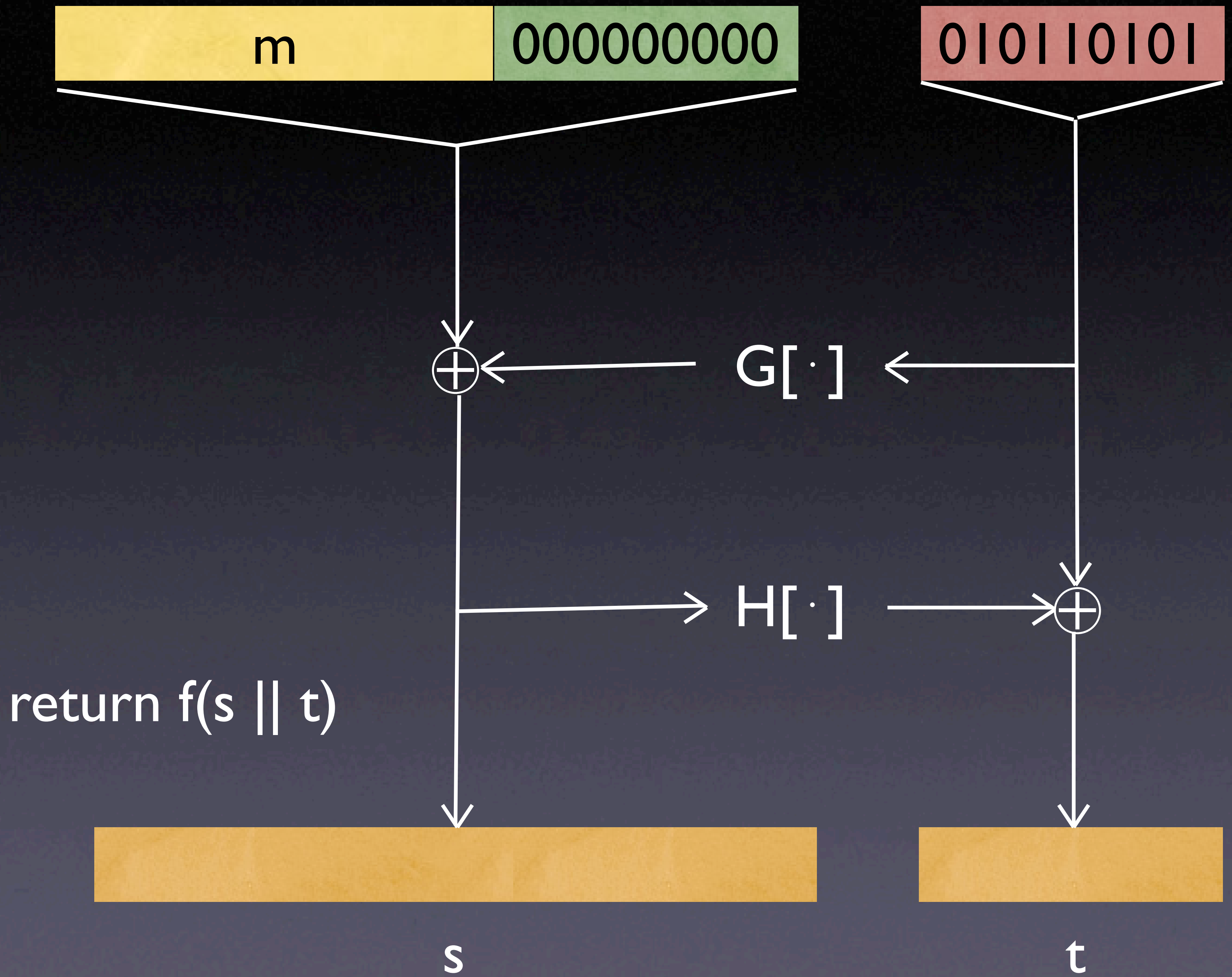
- Similar game to before:
  - Adversary given access to encryption and decryption oracles
  - Also given access to the random oracles G and H
  - Given the encryption of either m_0 or m_1, has to decide which it is

Break OAEP, you've broken the OWTP

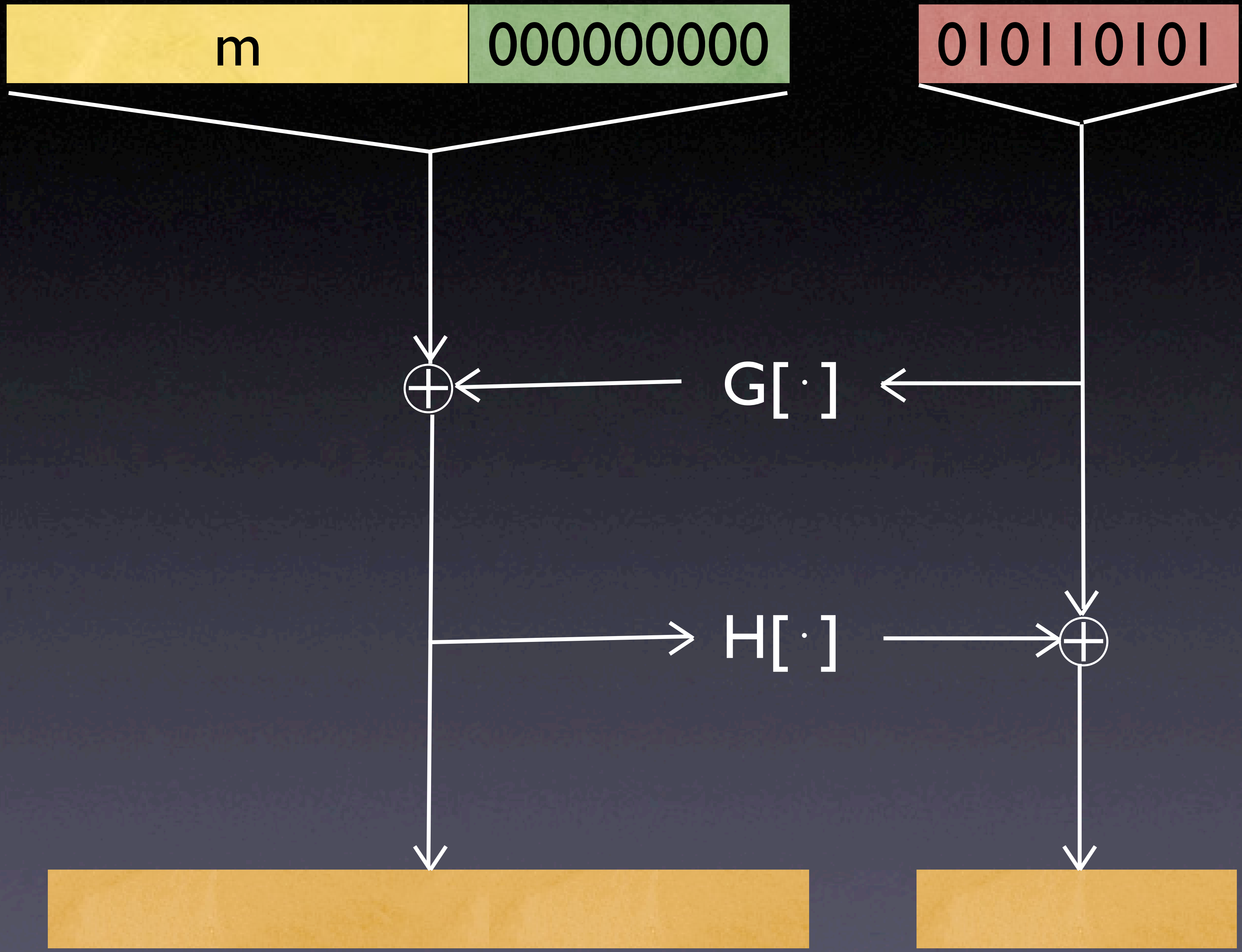
- Use the adversary that breaks OAEP to break the underlying one-way trapdoor permutation
- If the adversary can win at the m_0 or m_1 game, we can invert f (i.e. given a y, come up with x s.t. $f(x) = y$)

Adversary B(f, y)
// Wants to find x s.t. f(x) = y
Run A
    When A asks for G(x):
        See if G[x] exists, if so return it
        Generate G[x] at random, return it
    When A asks for H(x):
        See if H[x] exists, if so return it
        Generate H[x] at random, return it
...

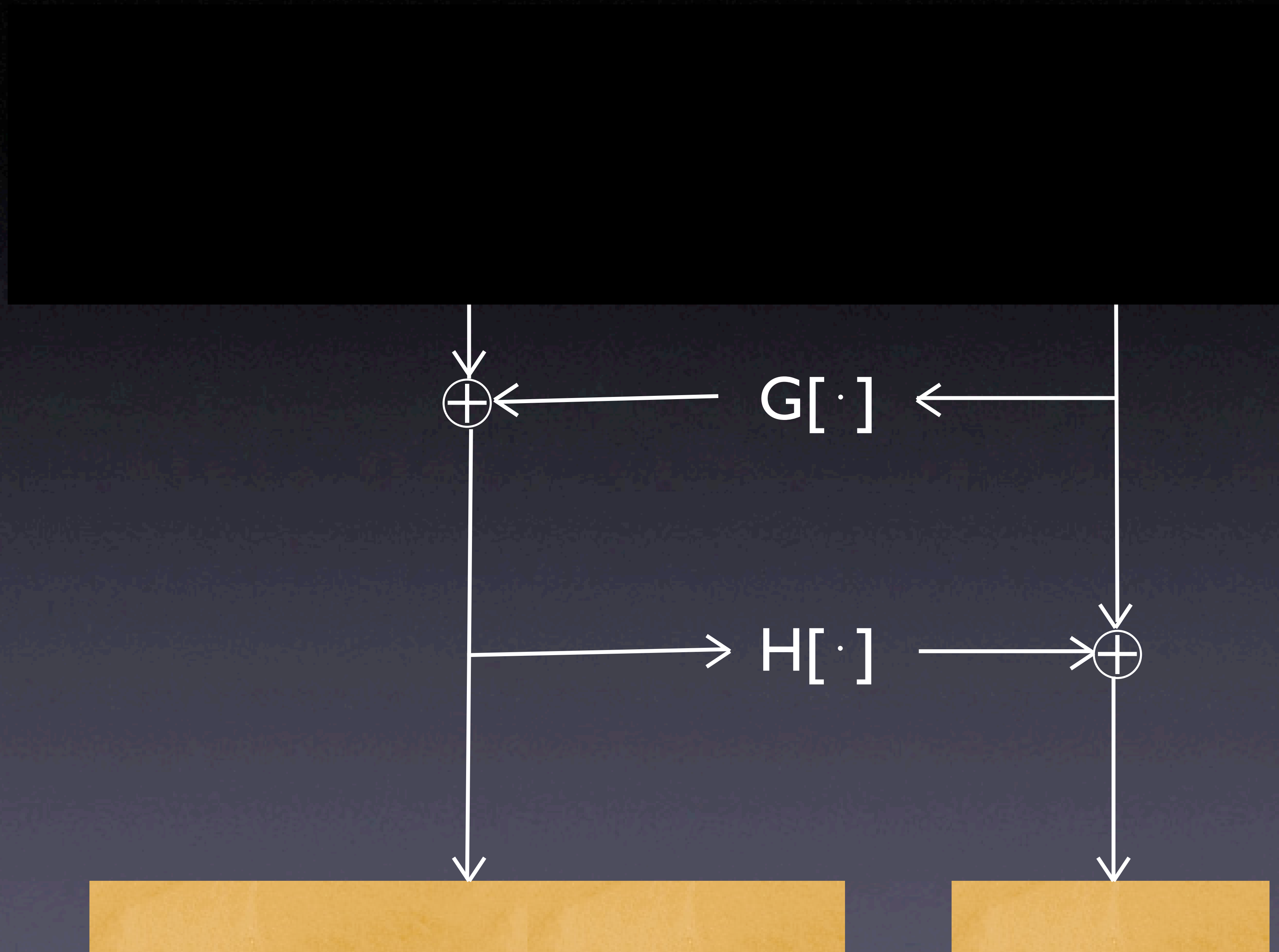
When A asks for E(m):



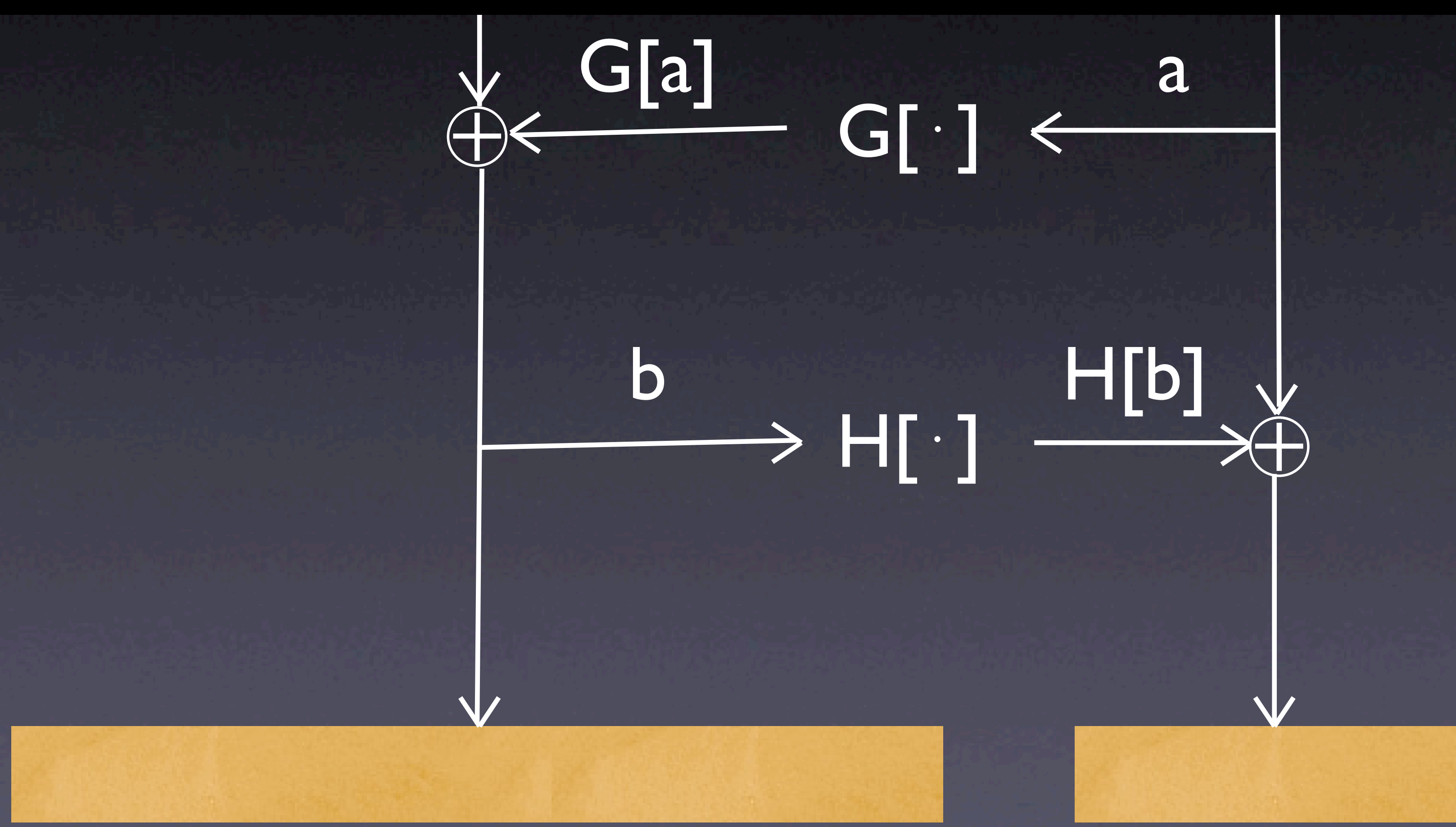
When A asks for D(c):



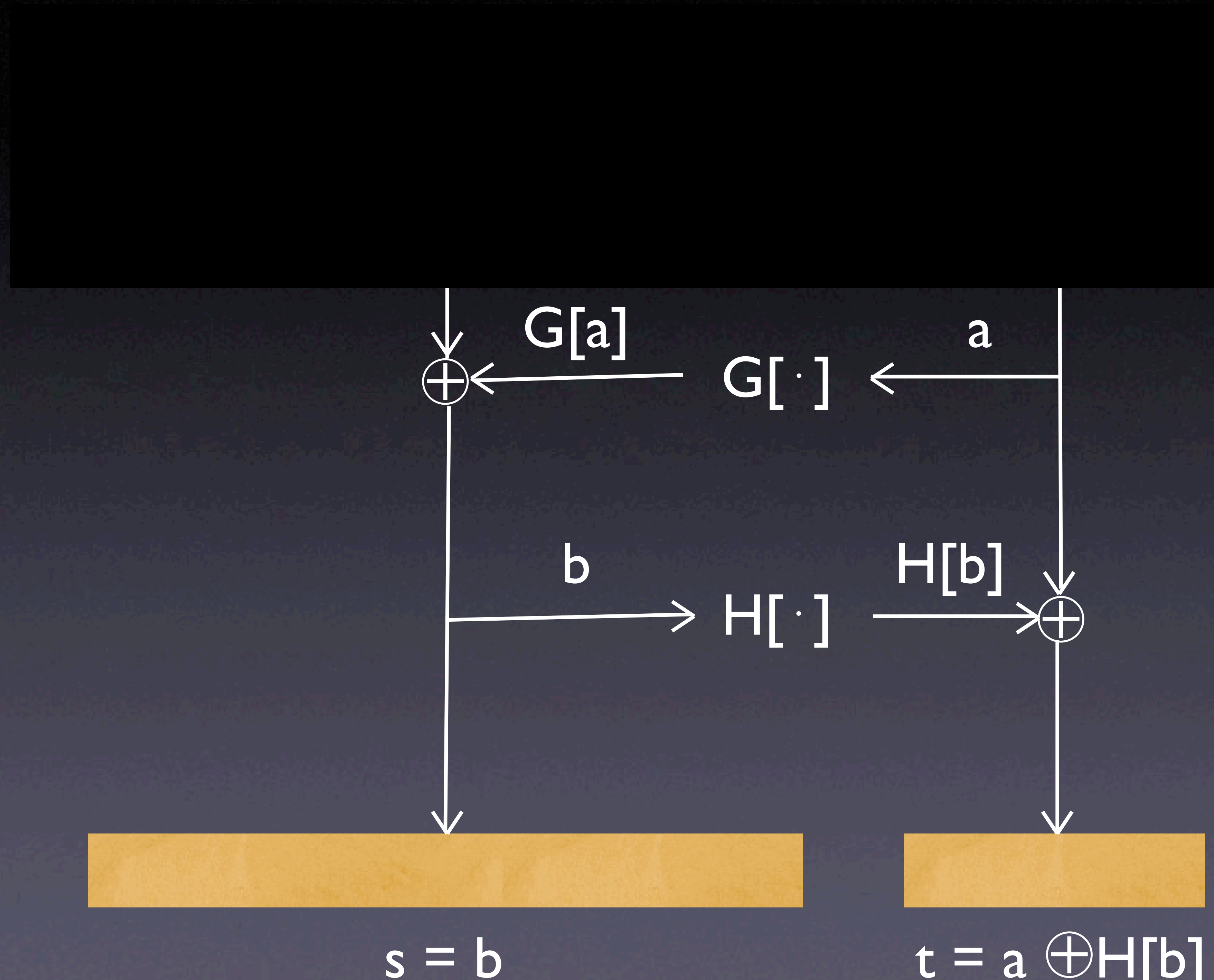
When A asks for D(c):



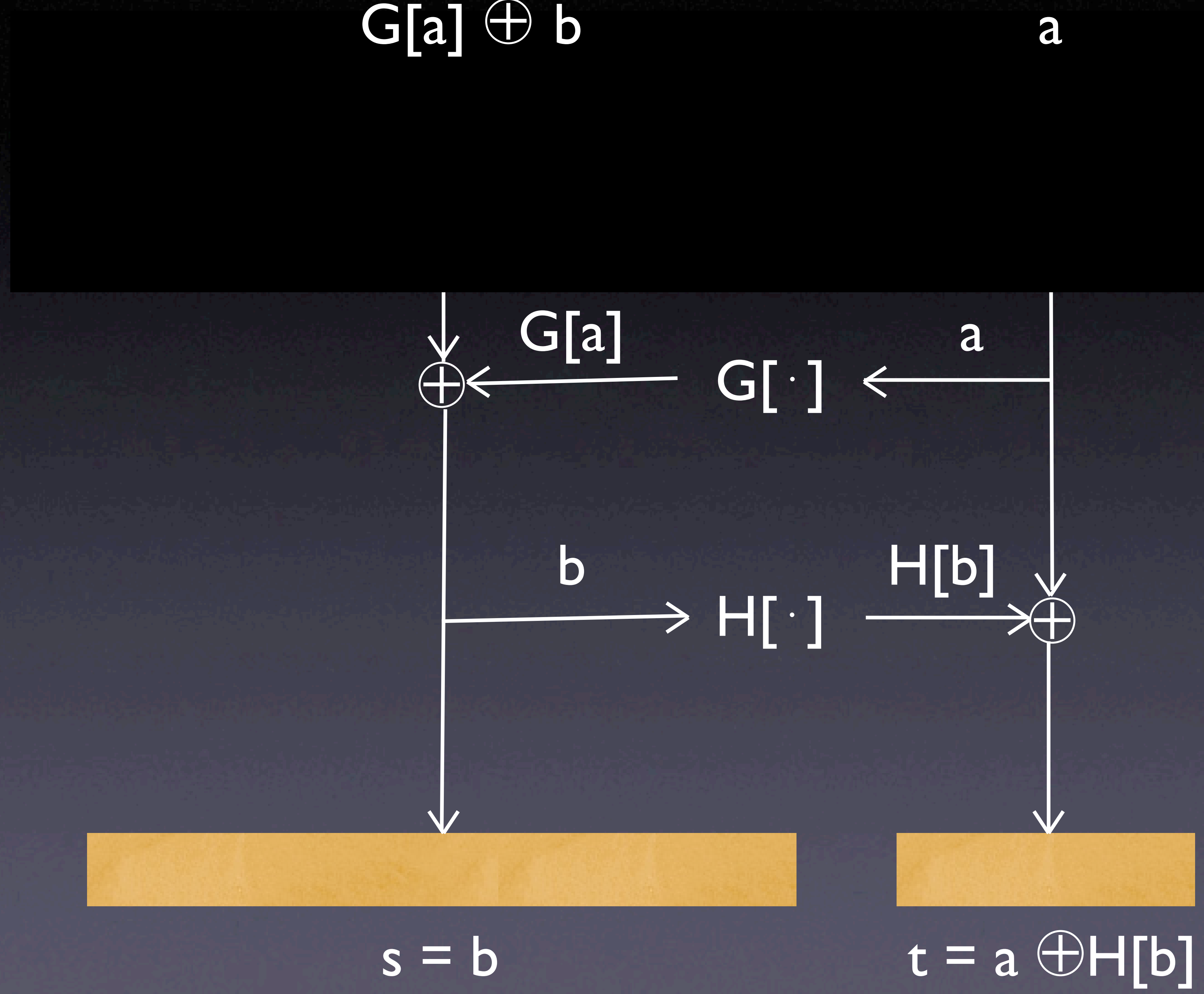
When A asks for D(c):



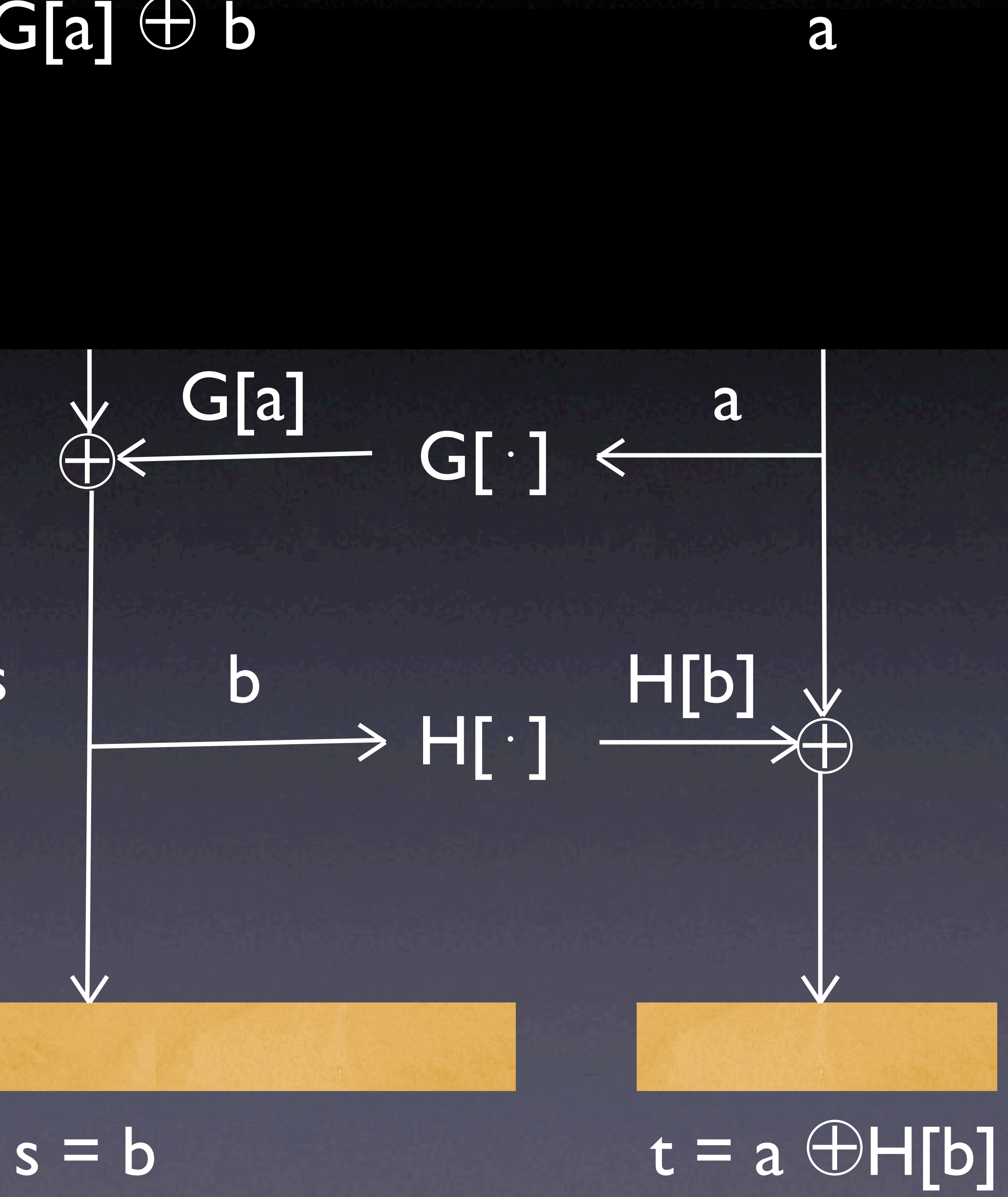
When A asks for D(c):



When A asks for D(c):



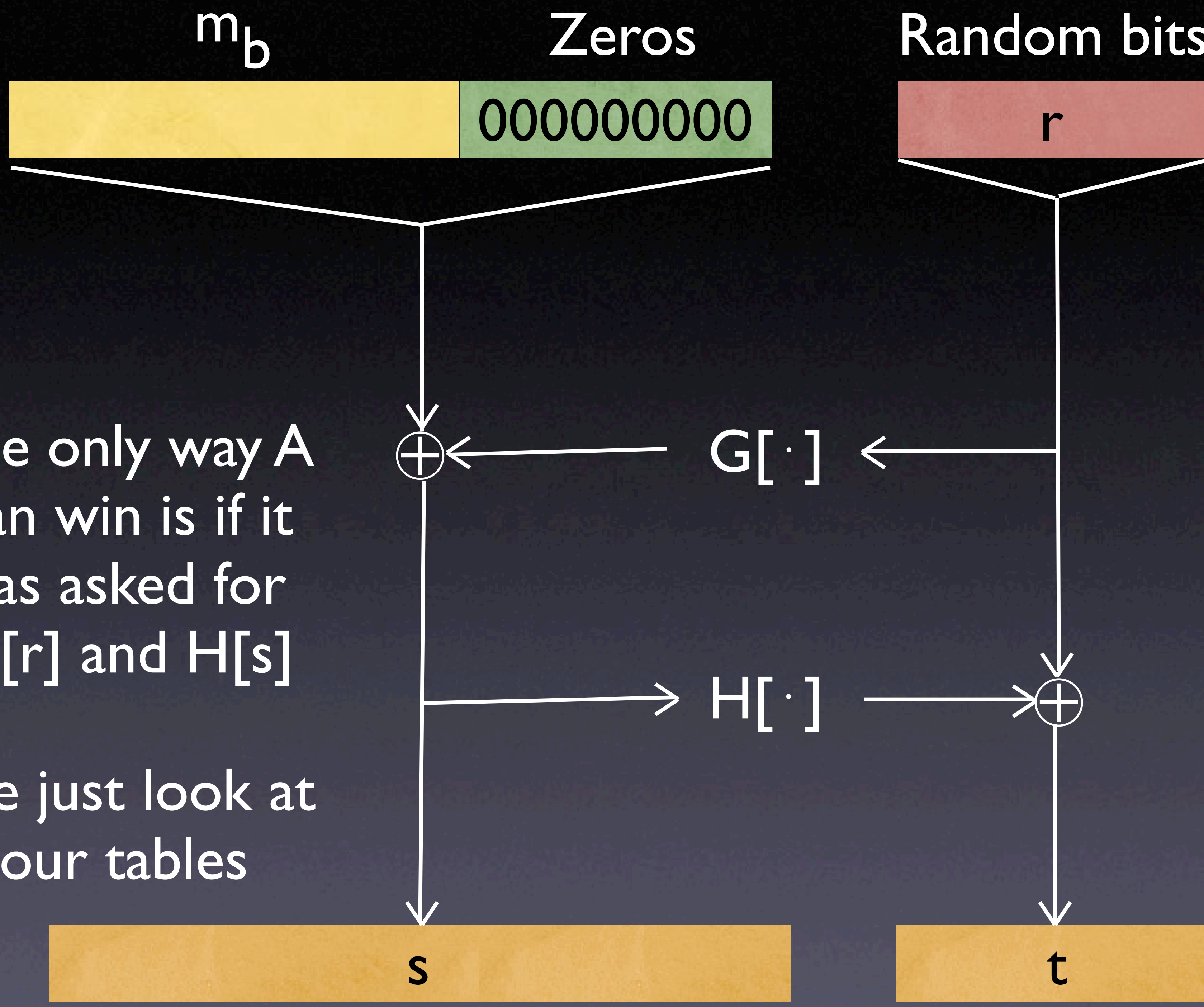
When A asks for D(c):



For index a of G[]
  For index b of H[]
    if $f(b \,||\, a \oplus H[b]) = c$
      if $G[a] \oplus b$ has Zeros
        return $G[a] \oplus b$
return $\perp$

A gives us m_0 and m_1

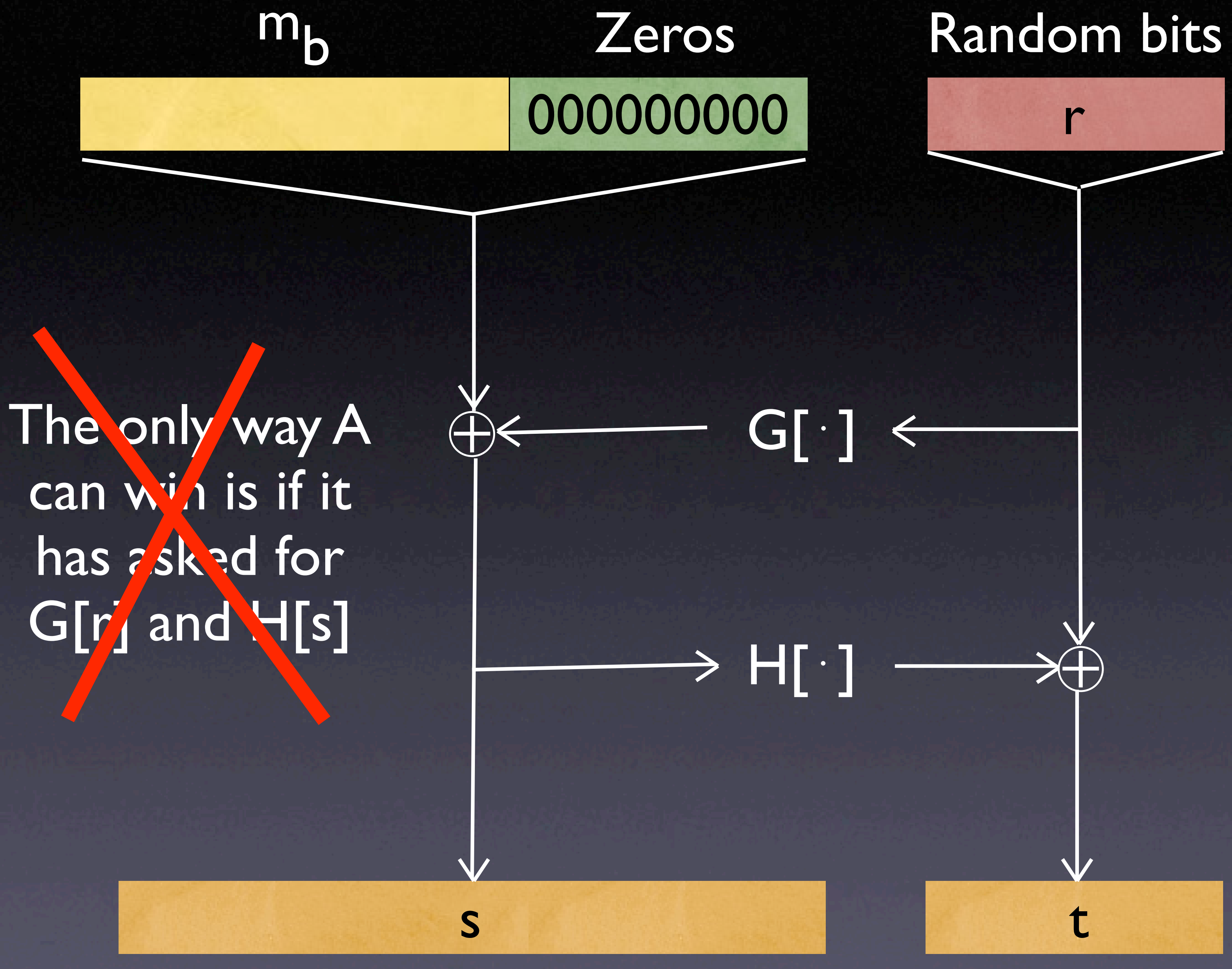
No matter what, we say that the encryption is y (remember that y is the thing we're trying to invert)



The only way A can win is if it has asked for G[r] and H[s]

We just look at our tables

$y = f(x) = f(s \,||\, t)$



m_b
Zeros
Random bits
000000000
r
G[·]
H[·]
s
t
The only way A can win is if it has asked for G[r] and H[s]

A Weird OWTP

- Given y you *can* compute the first few bits of x s.t. y = f(x)
- Given y you can compute some z s.t. x and w differ only in a few specific locations where y = f(x) and z = f(w) differ
- Don't know of any real examples, but can't rule it out

We Want CCA

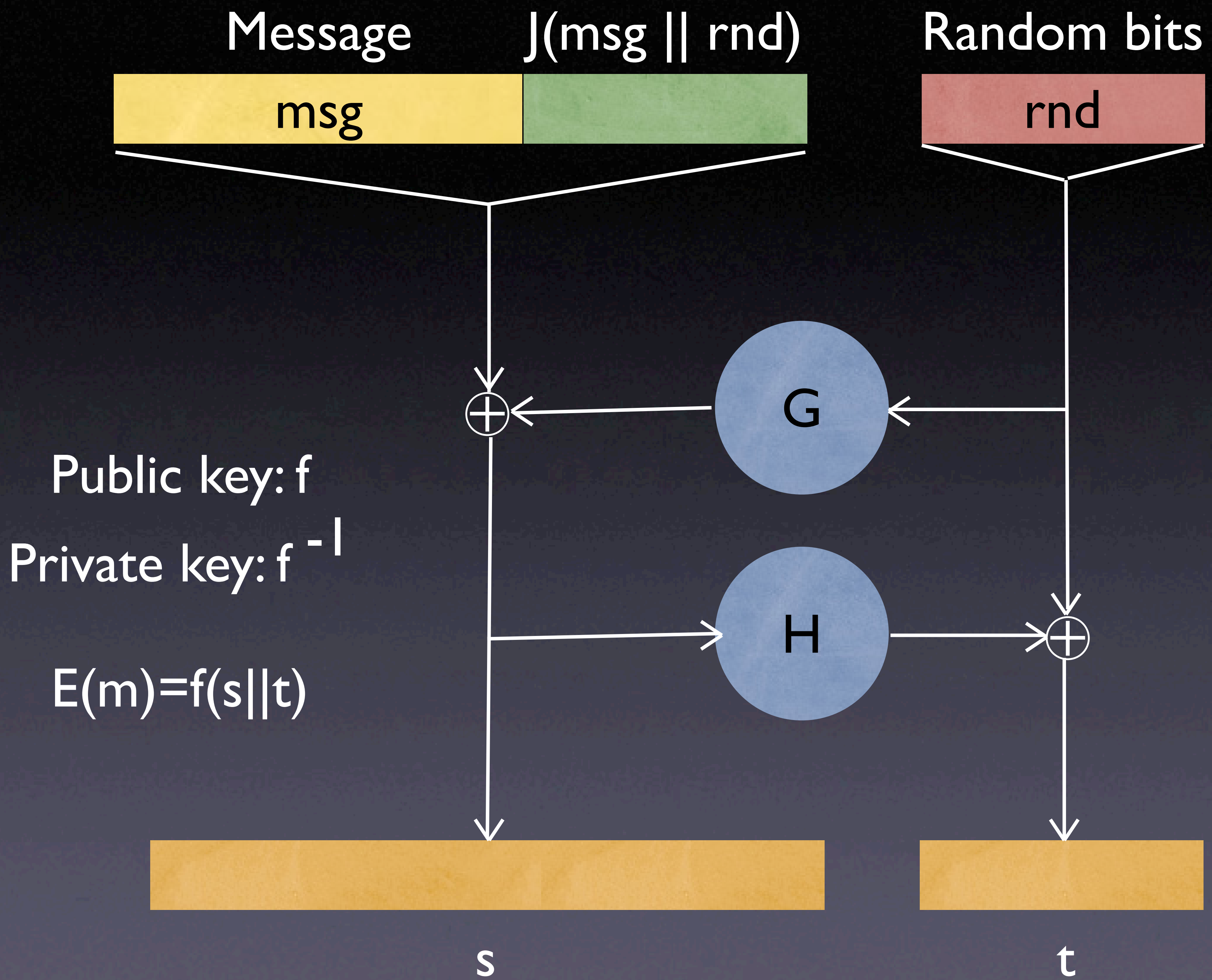
- OAEP paper proves that OAEP is plaintext aware (PA1)
- Few years later, another paper by Bellare et. al. show that:
  - Plaintext awareness implies CCA
  - This implies OAEP is IND-CCA

We Want CCA

- OAEP paper proves that OAEP is plaintext aware (PA1)
- Few years later, another paper by Bellare et. al. show that:
  - Plaintext awareness (PA2) implies CCA
  - This *does not imply* OAEP is IND-CCA

Some Good News

- OAEP is still secure when the OWTP is RSA (uses a special property of RSA)
- Easy to fix OAEP so that it works with *any* OWTP (OAEP+)
- For some OWTPs OAEP is overkill (SAEP)


Message
J(msg || rnd)
Random bits
msg
rnd
G
H
Public key: f
Private key: f -1
E(m)=f(s||t)
s
t

Lessons

- OAEP published in respected, peer-reviewed security conference by two top cryptographers
- PA→CCA paper published is respected, peer-reviewed security conference by same top cryptographer (and students)
- Bug not found until seven years later when Shoup tried to prove that OAEP was IND-CCA directly

Sources of Security Designs

- Commercial products
  - Truly revolutionary one million bit virtual matrix encryption

Sources of Security Designs

- Standards
  - Reviewed by other members of the standards committee
  - What if the standards committee doesn’t include any security people?

Sources of Security Design

- “The Literature”
  - Peer reviewed academic conferences and journals

Conferences

- Each program committee member given a stack of about 20 papers to review in a month
- Lead time to publication: 9 months

Journals

- A couple of reviewers given a couple of months to review one paper
- Lead time to publication: > 2 years

"The proof below spans more than 23 pages, and as much as I tried to simplify and to explain clearly, it is quite a pain to read. Frankly, I don't believe that anyone will ever go through the trouble of reading and verifying it."

Fermat’s Last Theorem

- Proof over 200 pages
- Subtle flaw found, able to be plugged before publication

Best Practice?

- Use what everyone else uses
  - At least people will be looking at it
- Still have to make sure that your implementation is secure

Our First Proof

- We want to prove that the following construction a is weakly unforgable MAC on variable length messages in the R.O.M:
  - $\text{ROMAC}_{k1,\,k2}(m) = f_{k1}(R(k2 \,||\, m))$

If f_{k1} is a weakly unforgable MAC on L bits and R is a random oracle with fixed L bit outputs then $ROMAC_{k1, k2}$ is a weakly unforgable MAC on variable length inputs.

Adversary given access to R and MAC and has to generate a valid new (m, t) pair

Given an adversary that forges ROMAC, come up with an adversary that forges f

Step 1: Run A
Step 2: Show how to answer A's queries
Step 3: Show how to use A's forgery of ROMAC to forge f