Cryptographic Hash Functions
Random Oracles

CS / ECE 407

Today's objectives

Introduce cryptographic hash function

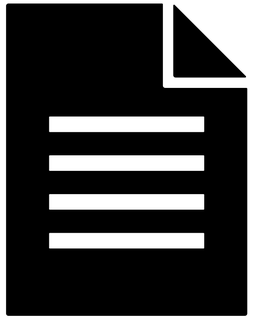
Understand the differences between hash functions and ciphers

Define the random oracle model

See some application of hash functions to MACs

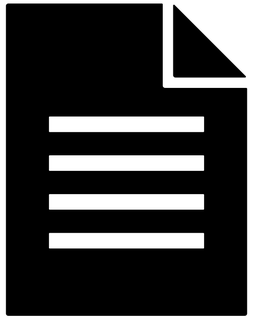
Alice

Bob



A.txt

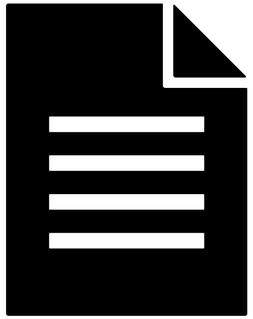
Are these files the same?



B.txt

Alice

Bob

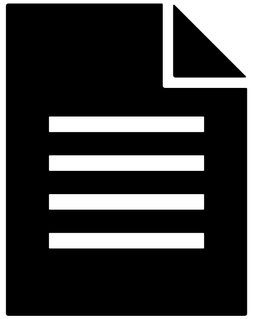


A.txt

1TB

Are these files the same?

Can we somehow compress the files to check efficiently?



B.txt

1TB

Modern Cryptographic approach to provable security

0) Define assumptions

1) Define Security

2) Specify a system

3) Prove that if assumptions hold, system is secure

There's a public function H, and on any input H returns a random output

Desired property for Random Oracle

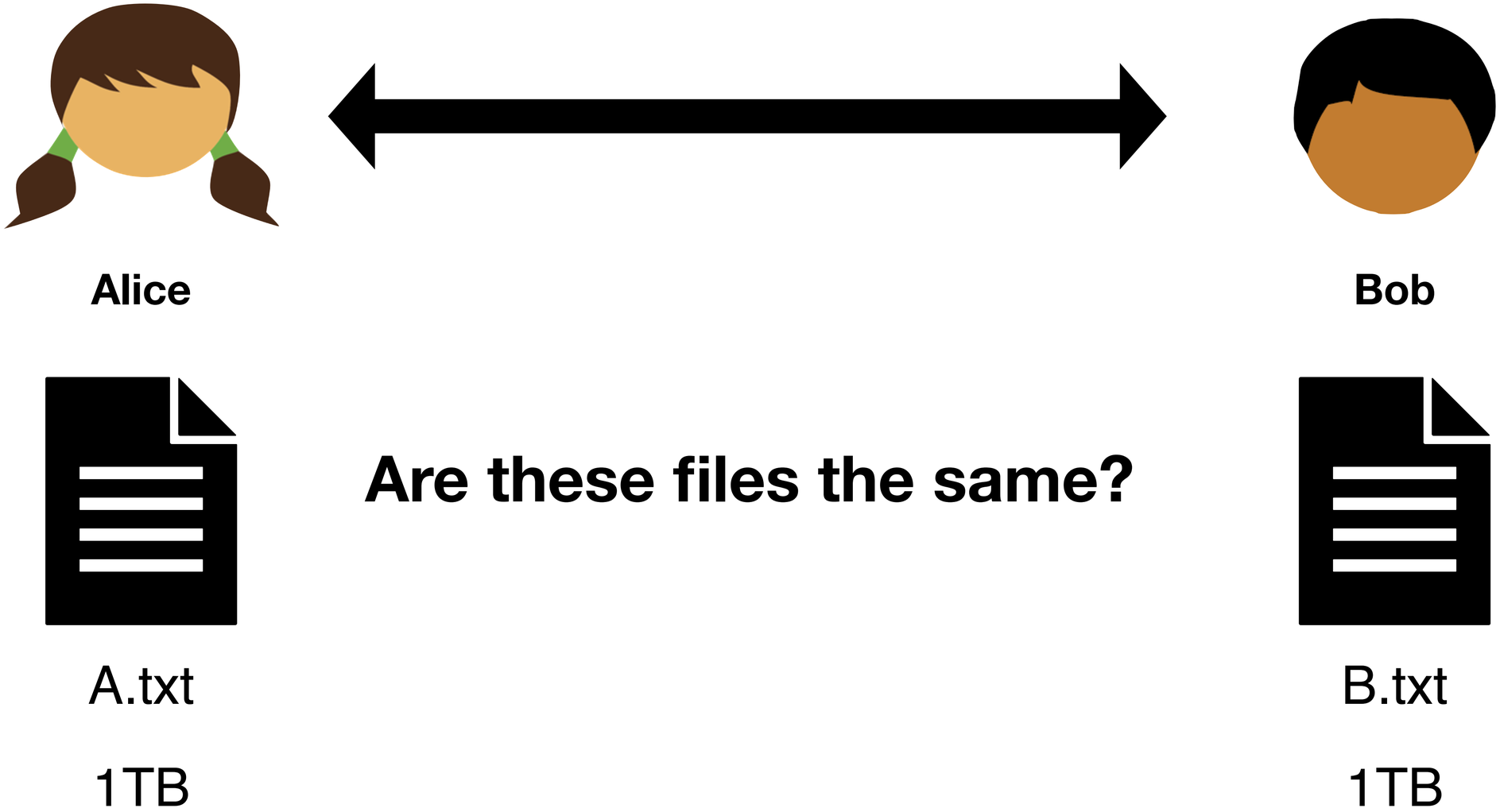
A function H : $\{0,1\}^* \rightarrow \{0,1\}^\lambda$ is a random oracle if the following indistinguishability holds

```
apply(x):
  return H(x)
```

$\overset{c}{\approx}$

```
D ← empty-dictionary

apply(x):
  if x is not in D:
    D[x] ←$ {0,1}^λ
  return D[x]
```



If H is RO, then hashing the files and comparing digests gives the right answer with overwhelming probability

Desired property for Random Oracle

A function H : $\{0,1\}^* \rightarrow \{0,1\}^\lambda$ is a random oracle if the following indistinguishability holds

```
apply(x):
  return H(x)
```

$\overset{c}{\approx}$

```
D ← empty-dictionary

apply(x):
  if x is not in D:
    D[x] ←$ {0,1}^λ
  return D[x]
```

```
k ←$ {0,1}^λ

apply(x):
  return F(k, x)
```

$\overset{c}{\approx}$

```
D ← empty-dictionary

apply(x):
  if x is not in D:
    D[x] ←$ {0,1}^λ
  return D[x]
```

Random Oracles are Practical: A Paradigm for Designing Efficient Protocols

MIHIR BELLARE* PHILLIP ROGAWAY†

Abstract

We argue that the random oracle model —where all parties have access to a public random oracle— provides a bridge between cryptographic theory and cryptographic practice. In the paradigm we suggest, a practical protocol P is produced by first devising and proving correct a protocol P^R for the random oracle model, and then replacing oracle accesses by the computation of an "appropriately chosen" function h. This paradigm yields protocols much more efficient than standard ones while retaining many of the advantages of provable security. We illustrate these gains for problems including encryption, signatures, and zero-knowledge proofs.

1 Introduction

Cryptographic theory has provided a potentially invaluable notion for cryptographic practice: the idea of provable security. Unfortunately, theoretical work often seems to gain provable security only at the cost of efficiency. This is due in part to the following. Theorists view certain primitives (e.g., one-way functions) as "basic" and build more powerful primitives (e.g., pseudo-random functions) out of them in inefficient ways; but in practice, powerful primitives are readily available and the so-called basic ones seem to be no easier to implement. In fact theorists deny themselves the capabilities of practical primitives which satisfy not only the strongest kinds of assumptions they like to make, but even have strengths which have not been defined or formalized.

In order to bring to practice some of the benefits of provable security, it makes sense to incorporate into our models objects which capture the properties that practical primitives really seem to possess, and view these objects as basic even if the assumptions about them are, from a theoretical point of view, very strong. This paper highlights the efficacy and potential of one such approach. The idea is a simple one: namely, provide all parties —good and bad alike— with access to a (public) random oracle; prove correct a protocol in this model; then replace the random oracle by an object like a hash function. We stress that the proof is in the random oracle model and the last step is heuristic in nature. It is a thesis of this paper that significant assurance benefits nonetheless remain.

The idea of such a paradigm builds on work of Goldreich, Goldwasser and Micali [20, 21] and Fiat-Shamir [14]. It is guided by many previous "unjustified" uses of hash functions. Finally, it incorporates viewpoints which, shared and verbally articulated by many members of our community, should be regarded as folklore. In this light, we view our contribution as follows. First, we raise the implicit philosophy behind the use of a random oracle to an explicitly articulated paradigm which we maintain brings significant benefits to practice. Second, we systematically apply the paradigm to diverse cryptographic problems to obtain efficient solutions. Third, we provide definitions and proofs to show that some of the previously "unjustified" uses of hash functions can find justification in the random oracle model. Finally, we suggest constructions of hash functions which we believe are appropriate to instantiate the random oracle. We proceed by describing the paradigm in further detail. For details on background and related work see Section 1.3.

* High Performance Computing and Communications, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA. e-mail: mihir@watson.ibm.com.

† PS LAN System Design, IBM Personal Software Products, 11400 Burnet Road, Austin, TX 78758, USA. e-mail: rogaway@austin.ibm.com.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

1st Conf.- Computer & Comm. Security '93-11/93 -VA,USA
© 1993 ACM 0-89791-629-8/93/0011...$1.50

There is no such thing as a random oracle!

But we still use them all the time

How does this relate to provable security?

Modern Cryptographic approach to provable security

0) Define assumptions

1) Define Security

2) Specify a system

3) Prove that if assumptions hold, system is secure

But random oracle “assumption” is always false, for any fixed function H

Modern Cryptographic approach to provable security

0) Define assumptions

1) Define Security

2) Specify a system

3) Prove that if assumptions hold, system is secure

But random oracle “assumption” is always false, for any fixed function H

Doesn’t this mean we can prove anything we want?

Random Oracle Model

Alice **Bob**

There's a public function black-box function H, and on any input H returns a random output

Random Oracle <u>Model</u>

Alice **Bob**

There's a public function black-box function H, and on any input H returns a random output

This is a bit like changing the parties' model of computation

before, parties were simply poly-bounded algorithms — we were in the **standard model**

In ROM, parties are poly-bounded computations *with access to global service* ***H***

Alice

Random Oracle <u>Model</u>

Random Oracle <u>Heuristic</u>

Bob

0) Define assumptions

1) Define Security

2) Specify a system <u>where parties have access to RO</u>

3) Prove that if assumptions hold, system is secure

Alice

Random Oracle <u>Model</u>

Random Oracle <u>Heuristic</u>

Bob

0) Define assumptions

1) Define Security

2) Specify a system <u>where parties have access to RO</u>

3) Prove that if assumptions hold, system is secure

4) Replace all instances of RO by concrete function H, and pray

Practitioner

Theoretician

“Almost all known attacks on the RO heuristic are contrived. A proof in the ROM is almost as good as a proof in the standard model!”

Practitioner

“Almost all known attacks on the RO heuristic are contrived. A proof in the ROM is almost as good as a proof in the standard model!”

Theoretician

“A proof in the ROM does not demonstrate security, but it might help understand what a proof in the standard model would look like.”

Message Authentication Codes

A MAC scheme is a pair of algorithms (KeyGen, tag) such that:

Real

```
k ← KeyGen()

get(m):
  return tag(k, m)

check(m, t):
  return tag(k, m) = t
```

$$\overset{c}{\approx}$$

Fake

```
k ← KeyGen()
S ← empty-set

get(m):
  t ← tag(k, m)
  S ← S ∪ {(m, t)}
  return t

check(m, t):
  return (m, t) ∈ S
```

Message Authentication Codes

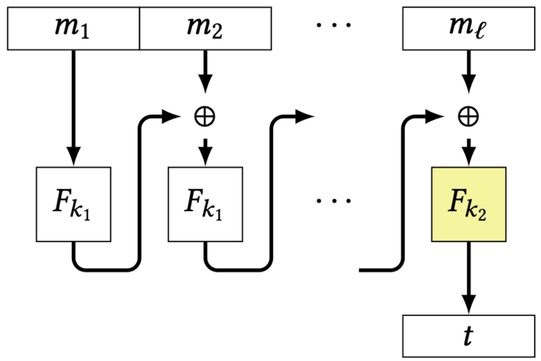
We saw that for PRF F with n-bit input, MAC on n-bit messages are trivial

But for longer messages, we had to be more clever

Let F be a PRF with in = out = λ. ECBC-MAC refers to the following scheme:

$\textsc{ecbcmac}^F\big((k_1, k_2), m_1 \cdots m_\ell\big)$:

$t := 0^\lambda$
for $i = 1$ to $\ell - 1$:
  $t := F(k_1, m_i \oplus t)$
return $F(k_2, m_\ell \oplus t)$



Message Authentication Codes

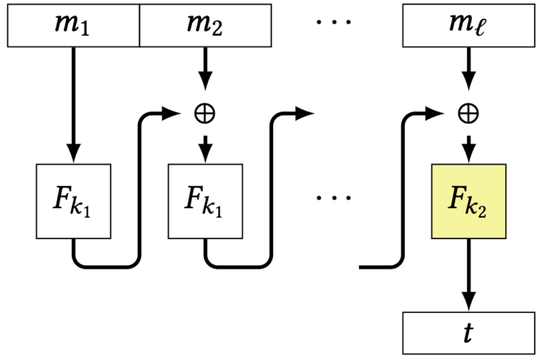
We saw that for PRF F with n-bit input, MAC on n-bit messages are trivial

But for longer messages, we had to be more clever

Let F be a PRF with in = out = λ. ECBC-MAC refers to the following scheme:

$\text{ECBCMAC}^F\big((k_1, k_2), m_1 \cdots m_\ell\big)$:

$t := 0^\lambda$
for $i = 1$ to $\ell - 1$:
  $t := F(k_1, m_i \oplus t)$
return $F(k_2, m_\ell \oplus t)$



HMAC:

```
KeyGen():
  K ←$ {0,1}^λ

tag(K, m):
  return H(K || m)
```

Today's objectives

Introduce cryptographic hash function

Understand the differences between hash functions and ciphers

Define the random oracle model

See some application of hash functions to MACs