
Abstracting Security in Distributed Computing Protocols

Charalampos Papamanthou

Brown University

Intel Research Berkeley

August 14th, 2008

Joint work with Petros Maniatis, Intel Research Berkeley

Scope

- Work is about distributed computing protocols such as
 - consistency protocols for replicated systems (e.g., BFT)
 - secure routing at the network layer and above (e.g., BGP)
 - secure in-network data analysis (e.g., aggregation protocols)
 - Security is essential for these protocols e.g.,
 - sign or encrypt a message
 - store a message in a secure log before sending it
 - Use of security primitives to ensure
 - consistency despite intrusions
 - fault tolerance in case of malicious behavior
 - privacy
-

Problem

- Protocols use certain security primitives implementations
 - RSA signatures
 - SHA-256 MACs
 - And the proofs of correctness are based on these...
 - No flexibility to the underlying application
 - What if one breaks RSA? Maybe use El-Gamal...redesign the protocol...redo the proofs?
 - System Heterogeneity
 - specialized hardware for signatures at machine A -> use sigs!
 - only enough resources for MACs at machine B -> no way!
 - Use exactly what I need
 - only one part of the protocol needs public verifiability!
 - no need to entirely employ expensive signatures
 - No different problem than using abstractions in general
 - We view it from the security perspective
-

Goals

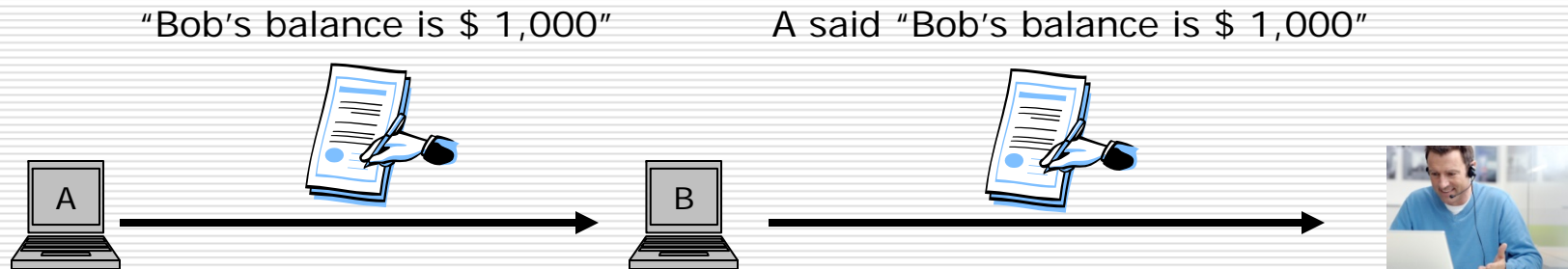
- **Abstract away the implementation details of security properties in distributed systems**
 - **authentication**: give me a proof that user A said “x”
 - **non-repudiation**: give me a proof that will allow me to prove to a third party that user A said “x”
 - **integrity**, i.e., give me a proof that the “x” I have is what its author said
 - Express abstractions with different implementations
 - Write a “smart” compiler that
 1. takes into account **(a)** execution environment, **(b)** computational power, **(c)** network infrastructure
 2. Chooses correctly by solving optimization problems
 - Overall goal: Better performance, more flexibility and still correct!
-

Concrete Example – Account Balance

- **Environment:**
 - Three banks A,B,C, at most one of which is faulty
 - Bob can only contact banks B,C. Bank A keeps his balance
 - Bank A says to B “Bob’s balance is \$1,000”
 - Bob wants to be able to prosecute A if in the future it presents a balance less than \$1,000
 - **Scenario:**
 - Bob contacts B
 - B should persuade Bob that A said his balance is \$1,000
 - **Abstraction:** A should send a “publicly verifiable message” to B (i.e., Bob and a judge should be able to verify it)
 - **Implementation:**
 - Digital signatures (public verifiability by default)
 - Message Authentication Codes (MACs) that support only point-to-point authentication
-

Implementation #1

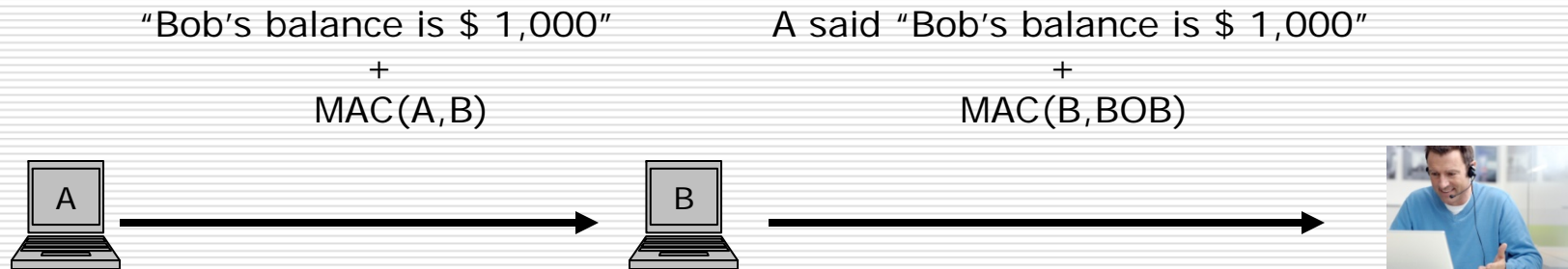
- Use digital signature (perfect fit since it supports public verifiability)



Bob **DOES NOT TRUST B**

Implementation #2

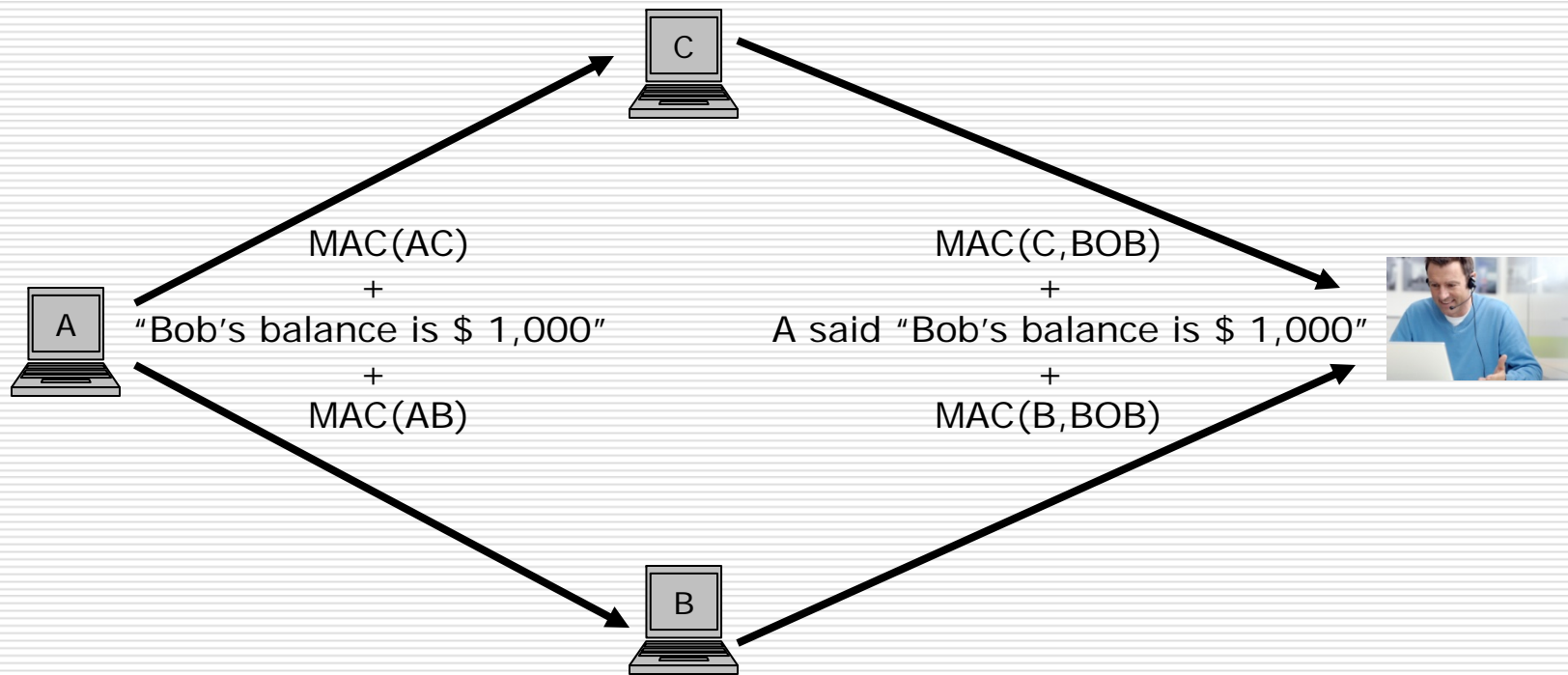
- Use MACs and some trust



Bob **TRUSTS** B

Implementation #3

- Use MACs and an auditor



At most one bank is faulty

Signatures VS MACs

	Digital Signatures	MACs
Protocol Complexity	2 messages	4 messages
CPU	1.5 x 2 ms (256-bit message) $O(1)$	0.00009 x 4 ms (256-bit message) $O(n)$
Bandwidth	1024 x 2 bits $O(n)$	256 x 4 bits $O(n)$

Summer 2008 at Intel Research

- Have come up with descriptions and semantics of security primitives
 - Have applied security abstractions to BFT (Byzantine Fault Tolerance) replication protocols
 - Two of the existing (different) solutions for 3-phase BFT
 - signatures
 - MACs
 - Have created an abstract specification of 3-phase BFT protocols
 - Have rewritten the proofs based on the abstract protocol
 - Have shown that both existing protocols satisfy our abstract specification
 - We are planning to create an implementation that uses abstractions to call certain security primitives implementation so that to optimize performance
-

Concrete Example 2 - Integrity

- ❑ **Environment:** Bob outsources his authenticated dictionary to Alice and keeps the digest of the dictionary.
 - ❑ **Scenario:** Bob wants to check if Alice still has his data after 5 days.
 - ❑ **Abstraction:** Alice needs to provide integrity proofs for all the elements of the authenticated dictionary
 - ❑ **Implementation:**
 - Either Alice computes a hash of the whole dictionary and send the whole dictionary over (requires high bandwidth and Bob have lots of storage)
 - Or Alice builds a Merkle tree on top of the dictionary and gives separate proofs for each element
 - ❑ Although counter-intuitive, the second solution might be better at some computational and network environments!
 - ❑ Same goal again: Automate the procedure!
-