Applied Cryptography and Network Security

William Garrison

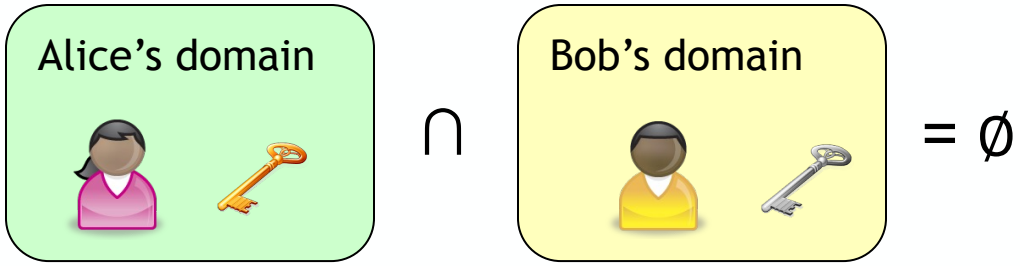
bill@cs.pitt.edu

6311 Sennott Square

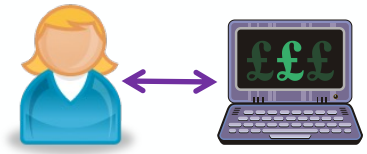
Secret sharing (threshold cryptography) and zero-knowledge proofs

University of Pittsburgh

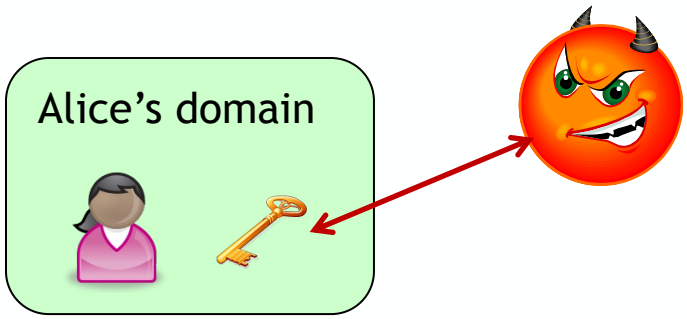
To date, we've (explicitly and implicitly) made a number of assumptions when discussion cryptosystems



Key Ownership



Device Utilization



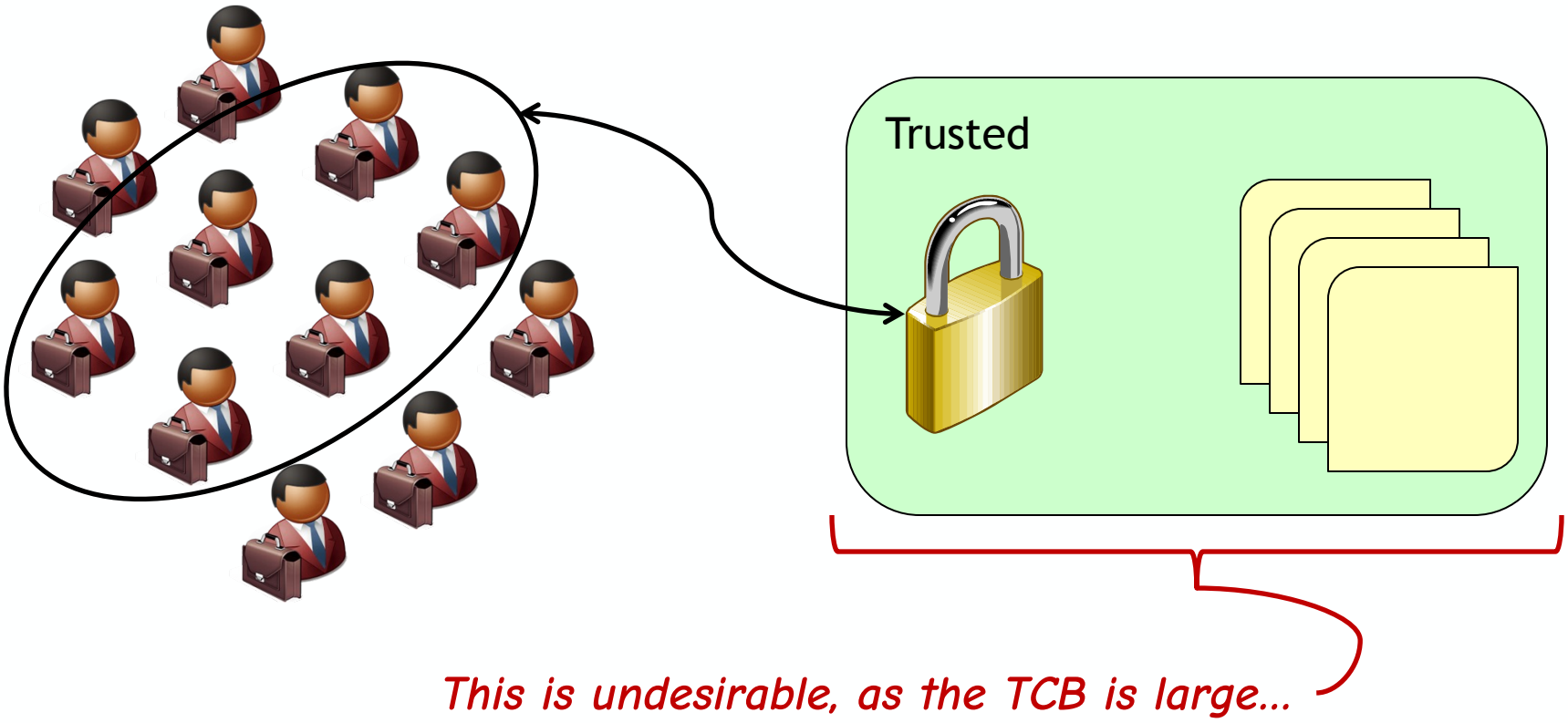
Key Compromise

We've seen several cryptosystems that meet our needs relative to these assumptions

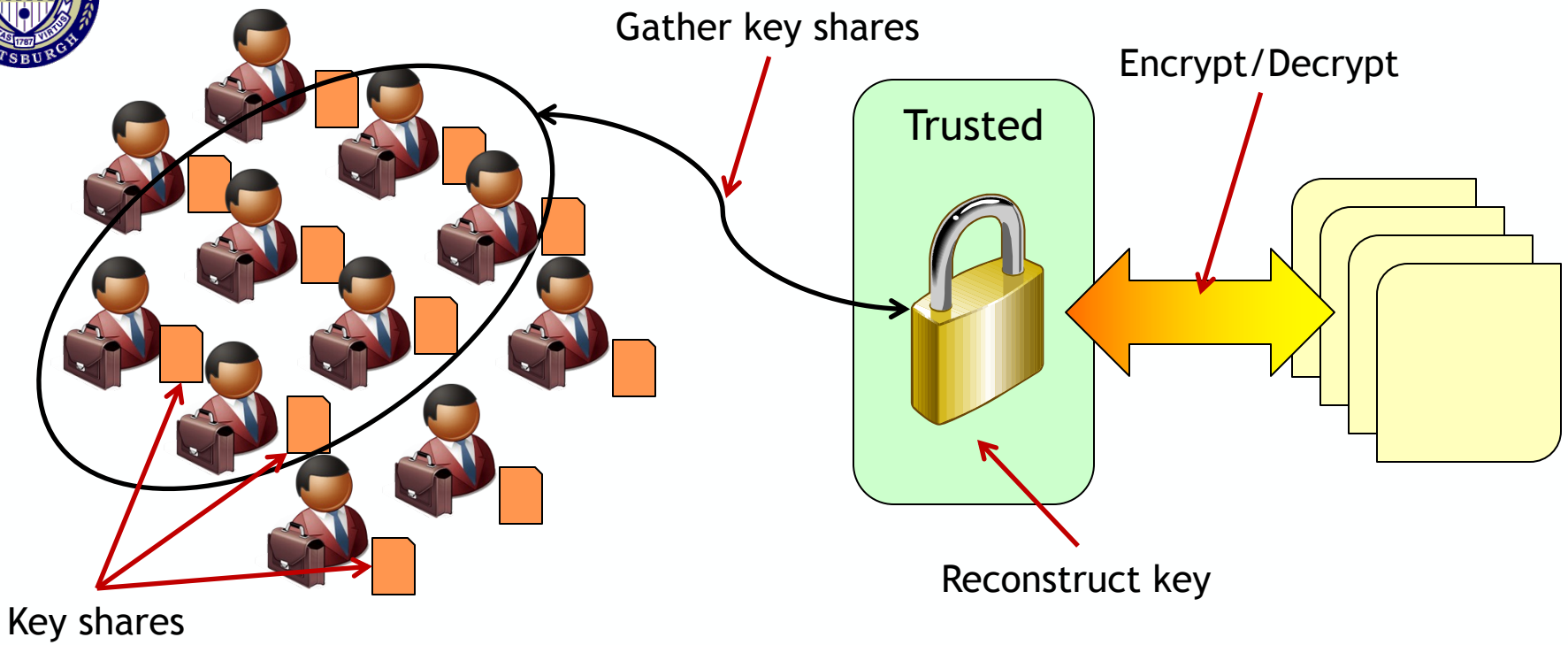
What happens if these assumptions are violated?

A motivating example…

Scenario: Eleven scientists are working together on an extremely sensitive joint project. To ensure that results are not tampered with, at least six scientists (i.e., a quorum) must be present in order to read or alter experimental results.



A better solution involves using a shared secret key



Goal: Given n users and a threshold $k \leq n$, divide a secret D into n pieces $D_1, D_2, \ldots, D_n$ such that:

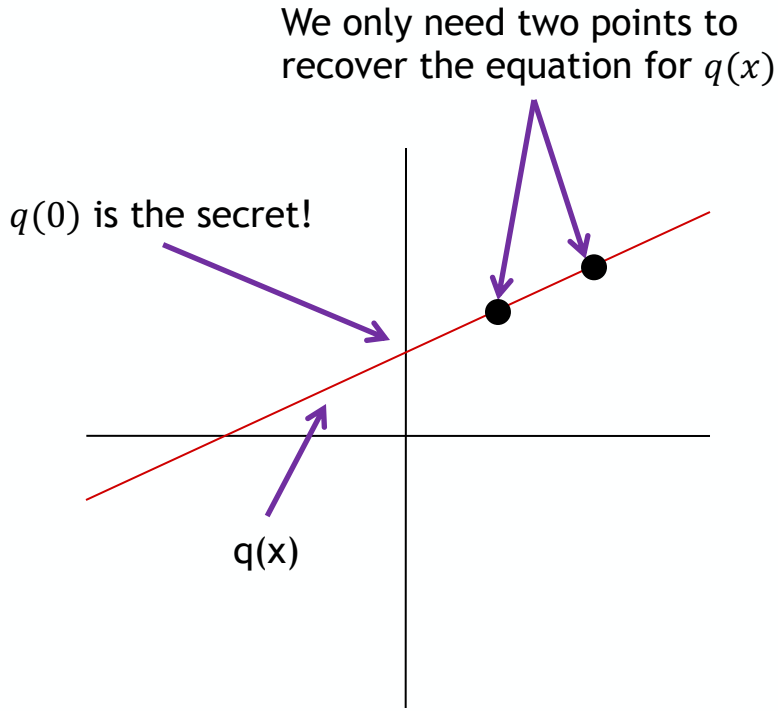
1. Any group of k or more users can jointly obtain the secret
2. Any group of $k-1$ or less users cannot obtain any information about the secret (i.e., all possible secrets remain equally likely)

Shamir's (k, n) secret sharing system allows us to accomplish this!

Paper: Adi Shamir, "How to Share a Secret," Communications of the ACM 22(11): 612-613, November 1979.

Intuitively, this scheme is quite simple

- Choose a random $(k-1)$-degree polynomial $q(x)$ whose root is D
- Each key share is a point on $q(x)$
  - $D_1 = q(1)$
  - $D_2 = q(2)$
  - …
  - $D_n = q(n)$
- Distribute $\{D_1, \ldots, D_n\}$ to participants
- $q(x)$ can be recovered by interpolation using any k shares
- $q(0) = D$



Toy Example: A line defines a $(2, n)$ secret sharing scheme

Details…

Rather than using real arithmetic, use modular arithmetic

- e.g., The set of integers modulo a large prime p, Z_p
- Note that p must be larger than both D and n

Otherwise we can't represent D!

Otherwise we will have collisions in our shares!

How do we randomly generate a $(k-1)$-degree polynomial $q(x)$ such that $q(0) = D$?

- Note that $q(x) = D + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$ is such a polynomial
- So, we just need to choose $a_1, \ldots, a_{k-1}$ at random!
- More precisely, choose $a_1, \ldots, a_{k-1}$ from the uniform distribution $[0, p)$

Note: All D_i values are computed modulo p

How is the secret recovered?

To recover D from a set of shares $\{D_1, \ldots, D_k\}$, solve the following system of k equations with k unknowns:

- $q(1) = D + a_1 1 + a_2 1^2 + \cdots + a_{k-1} 1^{k-1} = D_1$
- $q(2) = D + a_1 2 + a_2 2^2 + \cdots + a_{k-1} 2^{k-1} = D_2$
- …
- $q(k) = D + a_1 k + a_2 k^2 + \cdots + a_{k-1} k^{k-1} = D_k$

Unknowns: D, a_1, …, a_{k-1}

Note that any k shares can be used, not just $\{D_1, \ldots, D_k\}$

There exist $O(n \log^2 n)$ algorithms for polynomial interpolation

- In other words, the complexity of this operation is minimal
- For most (k, n) schemes, even a quadratic algorithm is fine

Is Shamir's scheme secure?

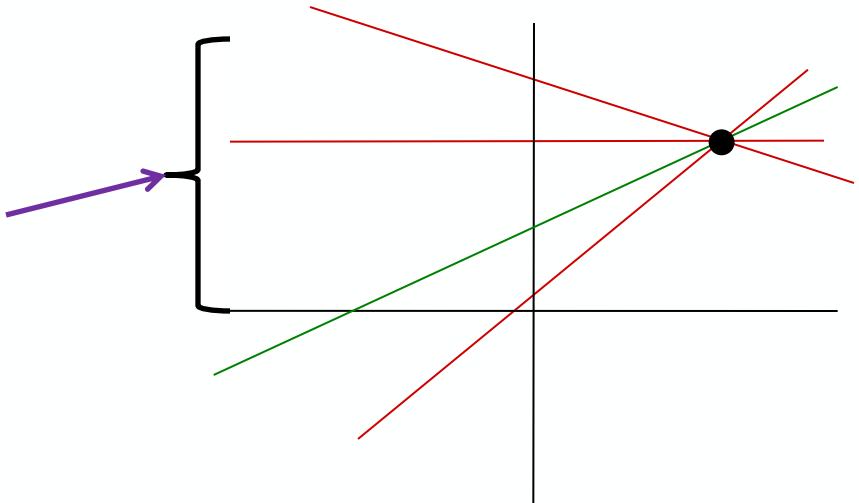
Recall: To prove security, we must show that any attacker who has $k-1$ shares thinks that all possible values of D are equally likely

Assume that the attacker has $k-1$ shares D_i

- For each candidate secret $D_c \in [0, p)$, exactly one $k-1$ degree polynomial $q'(x)$ can be constructed such that $q'(0) = D_c$ and $q'(i) = D_i$ for each known share
- So, all p possible polynomials are equally likely!

Degenerate Example: $(2, n)$ secret sharing

This entire space can be covered!



Shamir's scheme has many nice properties

The size of each share is less than or equal to the size of the secret D

- Why?

For a fixed k, shares can be dynamically added or deleted without affecting other shares!

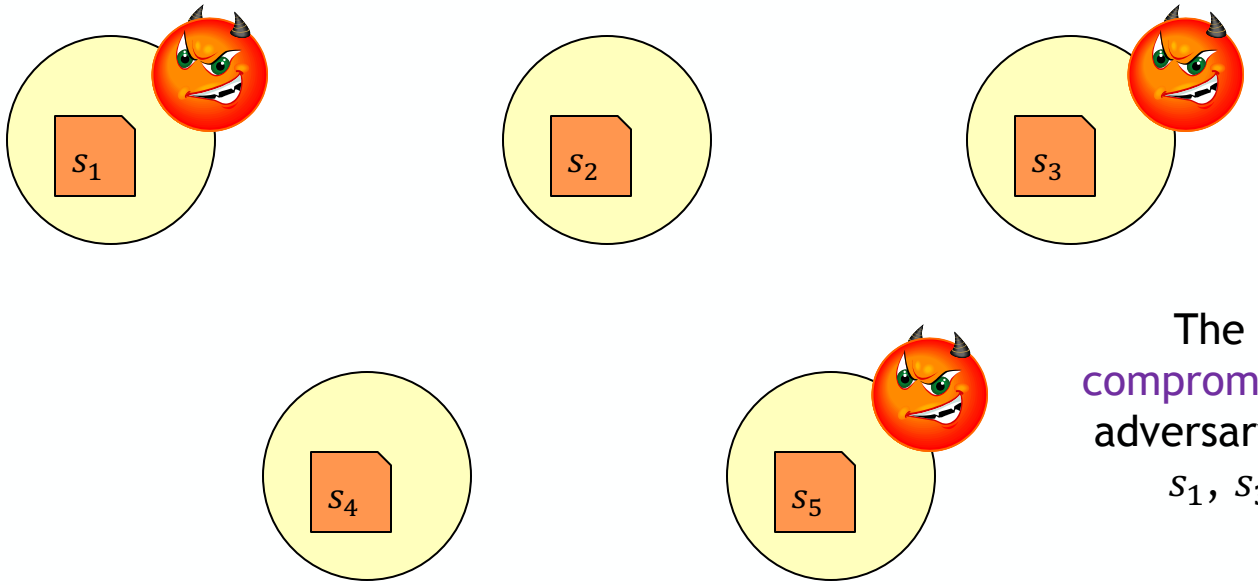
- Adding is easy---just compute new points along $q(x)$
- How do we delete a share?!?

We can develop a hierarchical scheme where the number of shares needed to find D depends upon the importance of principals

- ***Example:*** $(3, n)$ scheme
- 3 shares to president, 2 shares to each VP, 1 share to each board member
- Who can recover the secret?

This scheme is a building block that allows us to tradeoff security and reliability according to the values of k and n chosen

What happens as machines are compromised over time?



The system is compromised since the adversary has learned s_1, s_3, and s_5!

Example: A (3, 5) secret sharing system

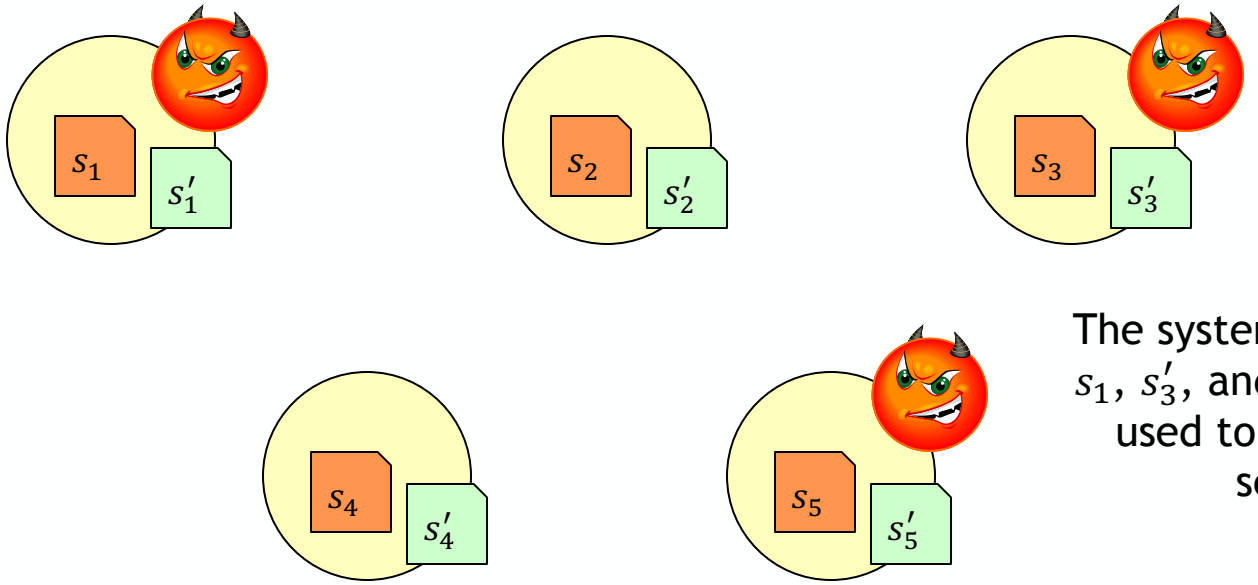
Consider a mobile adversary, capable of compromising nodes

- Over time, more and more nodes compromised
- The adversary learns all secrets stored on each compromised node

If 3 nodes are compromised, the secret is leaked!

- Note: Compromises do not need to be simultaneous!

Proactive secret sharing solves this problem!



The system is safe since s_1, s_3', and s_5' cannot be used to recover the secret!

Assumptions:

- Time proceeds in epochs (hours, days, weeks, etc)
- At the start of each epoch, machines can be rebooted into a safe state

Main idea:

- Update secret shares at the start of each epoch
- New shares should invalidate old shares!
- Secret should not change (despite new shares)
- Shares from new epoch cannot be used in conjunction with old shares

How can we do this?

During the setup phase, a trusted dealer

- Determines $q(x)$ and shares as in Shamir's scheme
- Shares sent out to participants

To evolve shares, each participant P_i

- Creates a random $(k-1)$-degree polynomial $\delta_i(x)$ s.t., $\delta_i(0) = 0$
- Send $u_{ij} = \delta_i(j) \pmod p$ to each P_j
- After receiving all u_{ji}, set $D_i' = D_i + (u_{1i} + \cdots + u_{ni}) \pmod p$

Note: After share evolution $q'(x) = q(x) + \delta(x)$

- $\delta(x) = \delta_1(x) + \cdots + \delta_n(x)$
- Since $\delta_i(0) = 0$ for each i, $\delta(0) = 0$
- So $q'(0) = q(0) + \delta(0) = D + 0 = D$

This share is independent of the old share

The secret is unchanged, despite new shares!

Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung, "Proactive Secret Sharing or: How to Cope with Perpetual Leakage," CRYPTO 1995.

What are the properties of this PSS scheme?

The authors define security in terms of two important properties

Robustness: The new shares computed at the end of the update phase correspond to the secret D (i.e., any subset of k new shares can be used to interpolate the secret D).

Informally, we get this because $\delta(0) = 0$

Secrecy: An adversary that at any time period knows no more than $k - 1$ shares learns nothing about the secret.

Note that old shares become obsolete, since from the adversary's perspective, all possible update polynomials are equally likely. Then, within a single epoch, the proof follows from the secrecy of the Shamir scheme.

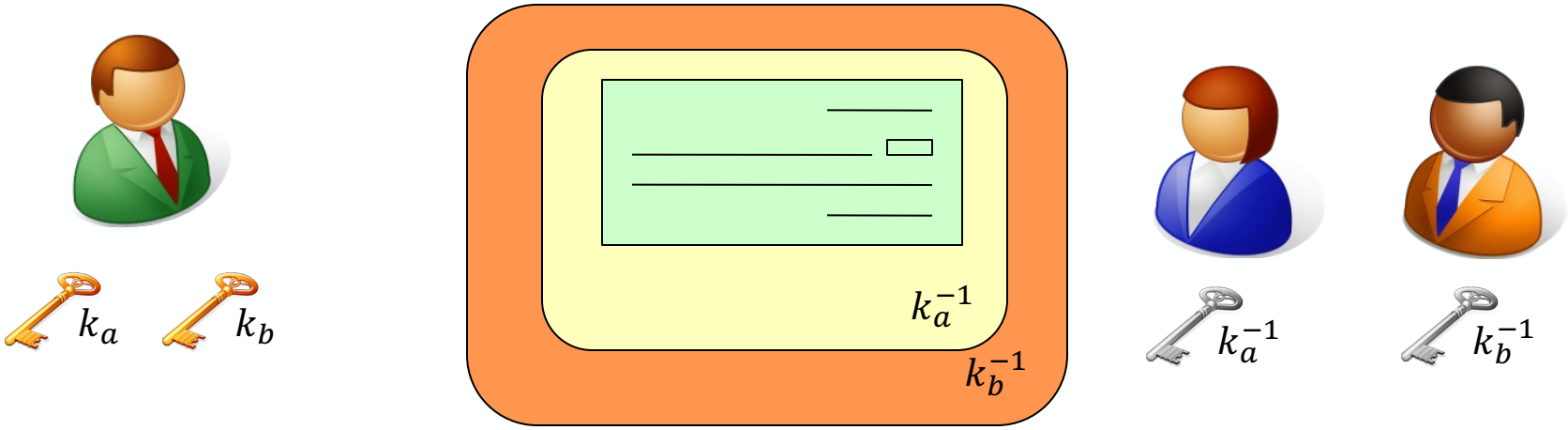
Certain distributed computing problems cannot be solved by using shared symmetric keys

Problem 1: Suppose that a company issues electronic checks, but requires the signatures of at least 3 executives to be considered valid.

Problem 2: A user wants to send an encrypted message to an organization. However, for accountability, the users wants at least 2 members of that organization to view the message together.

Solving these types of problems requires a public key cryptosystem where encryption and/or decryption requires cooperation!

A Straw-Man Approach



Require multiple digital signatures on the message!

Unfortunately, this solution has several pitfalls

- Verifying k signatures is a waste of time
- Message size is proportional to k
- Does not allow for group anonymity
- Doesn't work for decryption case

A better solution would be to define threshold variants of cryptosystems such as RSA

Main idea: Split the RSA decryption exponent d into shares using Shamir's secret sharing approach

How does the protocol work?

- Given a message m, each signer can compute a partial signature of m using his or her share of the decryption exponent
- These partial signatures can be multiplied together
- *Result:* A signature on m

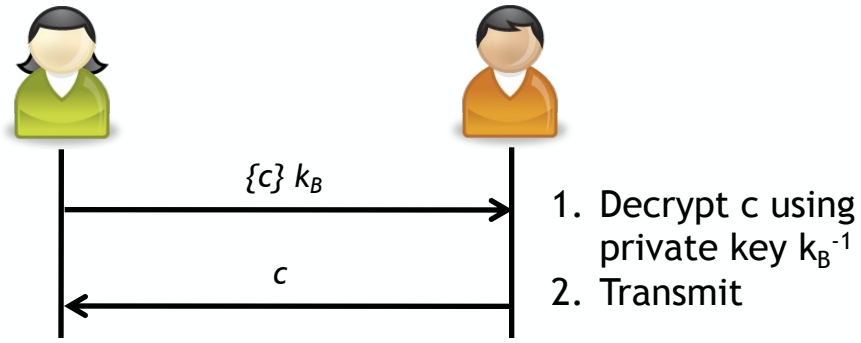
This protocol has a rigorous proof of security based upon standard cryptographic hardness assumptions

Want more details? See: Victor Shoup, "Practical Threshold Signatures," Proceedings of EuroCrypt 2000.

Note that public key cryptography allows us to prove knowledge of a secret without revealing that secret

Example: Decrypting a challenge

1. Pick challenge c at random
2. Encrypt c using Bob's public key k_B
3. Transmit

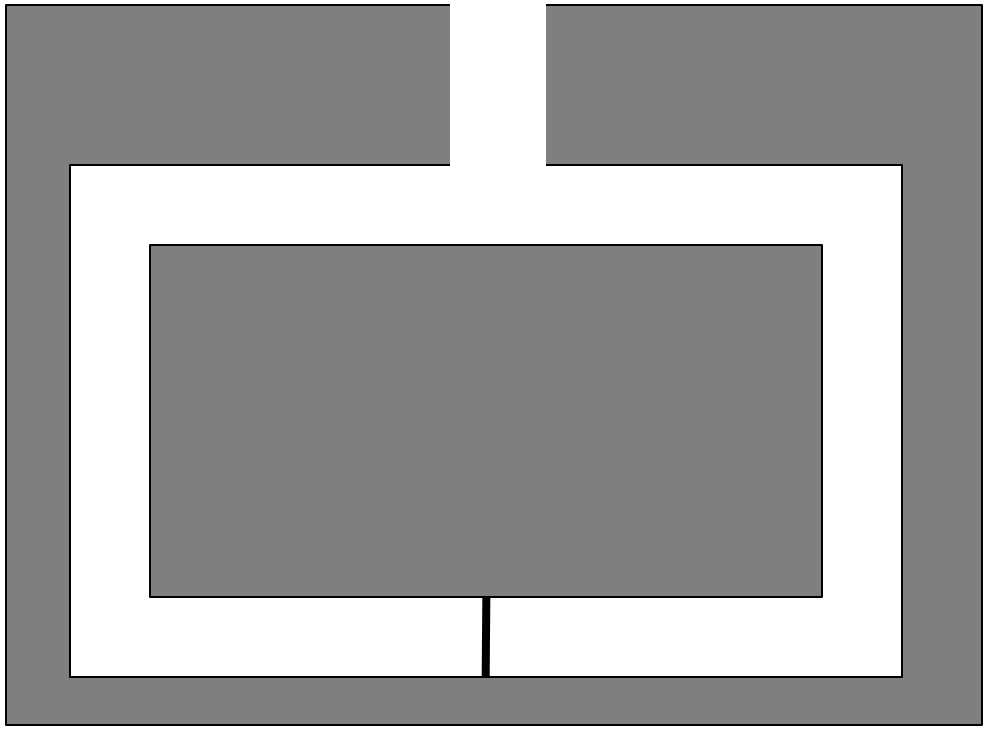


Note: Revealing the challenge, c, does not leak information about the private key k_B^{-1}, yet Alice is (correctly) convinced that Bob knows k_B^{-1}

This type of protocol is called a zero-knowledge protocol

Zero-knowledge proofs are easy in the abstract: Ali Baba's "open sesame"

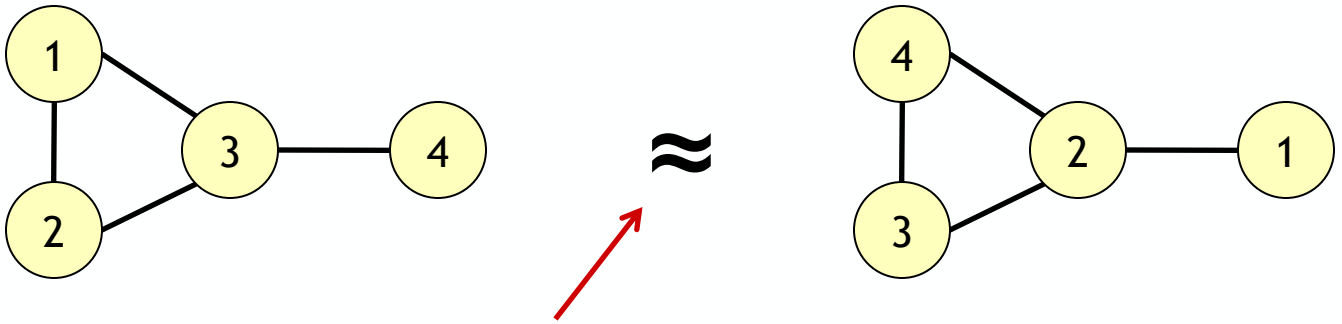
Example: The secret cave



Note: To ensure correctness, this "protocol" needs to be run multiple times (Why?)

We can construct a more realistic ZK system based on the (hard) problem of determining graph isomorphism

Informally, two graphs are isomorphic if the only difference between them is the names of their nodes



Via the permutation π = {4, 3, 2, 1}

Determining whether two graphs are isomorphic is in NP, and the best-known algorithm (as of 2017) is $2^{O((\log n)^c)}$ for some constant c (quasipolynomial). This means that if the graphs are large, solving this problem will take noticeably longer than checking a claimed solution.

- Much smaller imbalance than prior to 2017…

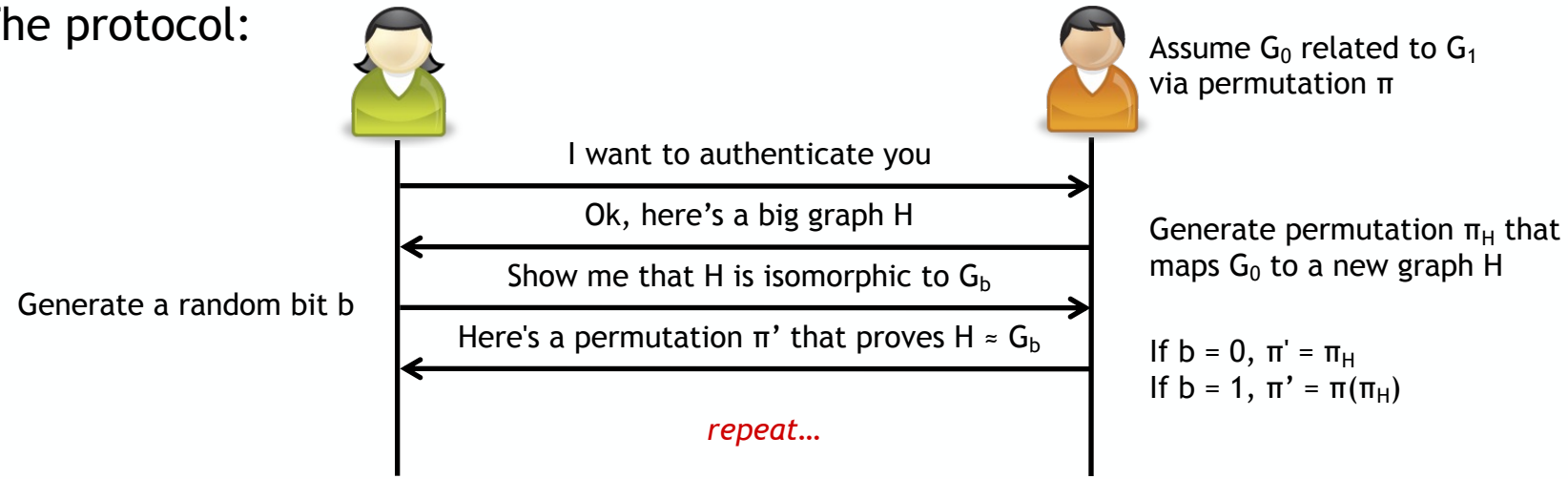
Authenticating via Graph Isomorphism

Our protocol has fairly simple parameters

- Public key: Two (big) isomorphic graphs G_0 and G_1
- Private key: The permutation mapping $G_0 \rightarrow G_1$

How do we find these efficiently?

The protocol:



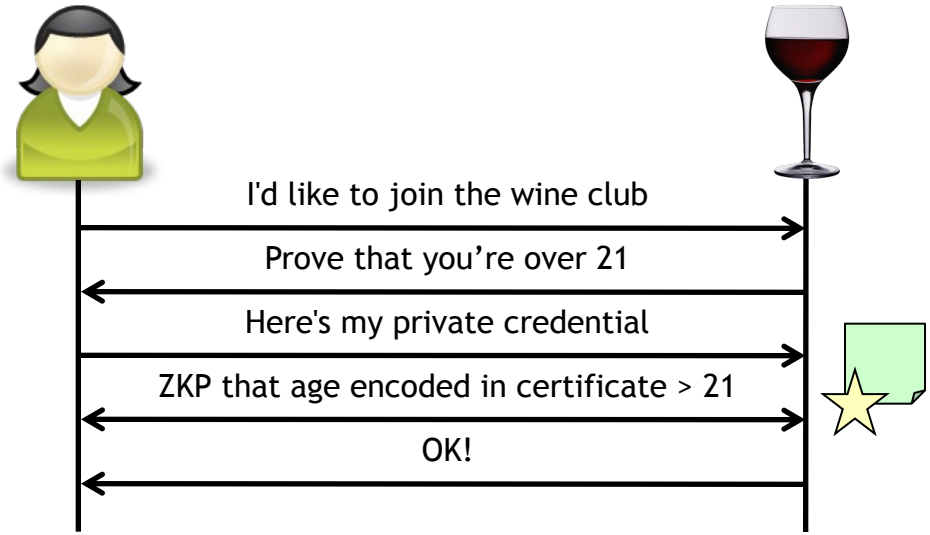
Why does this work?

- Answering this once means that Bob knows (at least) the permutation mapping from G_b to H.
- Doing this m times means that Bob knows the mapping between G_0 and G_1 with probability $1 - 0.5^m$
- Note that this leaks no information regarding the permutation π (Why?)

Zero knowledge proofs of knowledge can be used to solve a variety of interesting authorization problems

Private/Anonymous credential systems allow users to prove that they have certain attributes without actually revealing these attributes

Example: Purchasing wine over the Internet



The private credential scheme proposed by Stefan Brands enables many types of attribute properties to be checked in a zero-knowledge fashion

Conclusions

Threshold systems are useful when trying to secure distributed systems

Interesting properties:

- Knowing k shares allows a secret to be reconstructed
- Knowing even $k - 1$ shares provides no information

Extensions to this basic model have been proposed to allow

- Share evolution
- Tolerance of an untrusted “dealer”
- Shared signatures
- …

Zero knowledge proof systems have many interesting applications

Next time: Quantum computing