THE UNIVERSITY OF BRITISH COLUMBIA

Key Management

EECE 412

Session 7

Copyright © 2004 Konstantin Beznosov

Kerckhoff's Principle

"The security of a cryptosystem must not depend on keeping secret the crypto-algorithm. The security depends only on keeping secret the key"

Auguste Kerckhoff von Nieuwenhof

Dutch linguist

1883

Outline

- Key exchange
  - Session vs. interchange keys
  - Classical, public key methods
- Cryptographic key infrastructure
  - Certificates
- Quantum key distribution

Notation

- $X \rightarrow Y : \{ Z \,||\, W \} k_{X,Y}$
  - *X* sends *Y* the message produced by concatenating *Z* and *W* enciphered by key $k_{X,Y}$, which is shared by users *X* and *Y*
- $A \rightarrow T : \{ Z \} k_A \,||\, \{ W \} k_{A,T}$
  - *A* sends *T* a message consisting of the concatenation of *Z* enciphered using k_A, *A*'s key, and *W* enciphered using $k_{A,T}$, the key shared by *A* and *T*
- r_1, r_2 nonces ("nonrepeating" random numbers)

Session, Interchange Keys

- Alice wants to send a message *m* to Bob
  - Assume public key encryption
  - Alice generates a random cryptographic key k_s and uses it to encipher *m*
    - To be used for this message *only*
    - Called a *session key*
  - She enciphers k_s with Bob's public key k_B
    - k_B enciphers all session keys Alice uses to communicate with Bob
    - Called an interchange *key*
  - Alice sends { *m* } k_s { k_s } k_B
- Benefits?

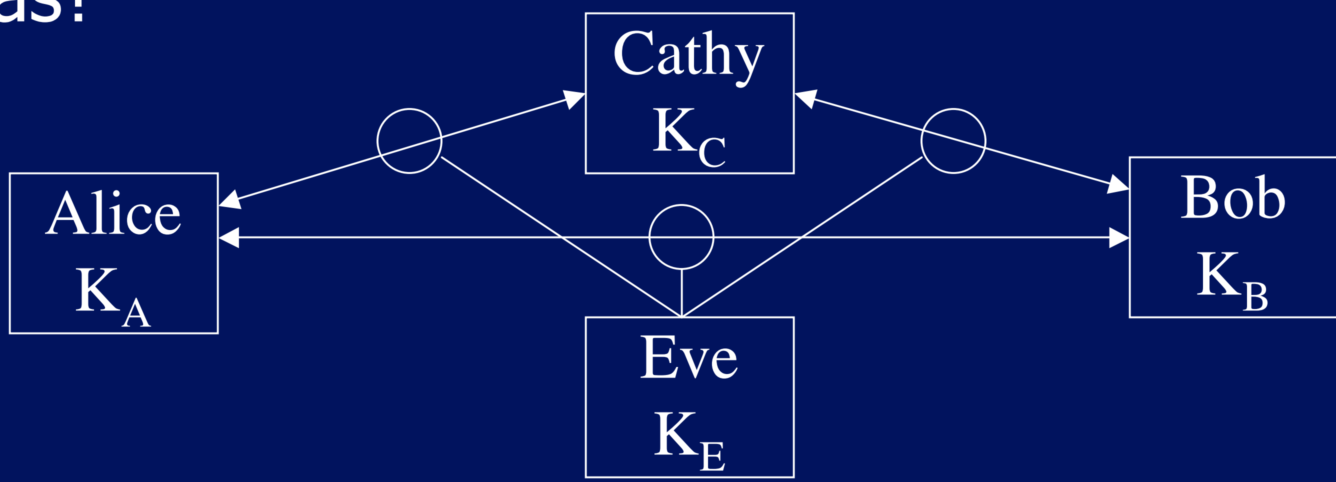
Key Exchange Algorithms

Goal: Alice, Bob get shared key

- Key cannot be sent in clear
- Alice, Bob may trust third party
- All cryptosystems, protocols publicly known

Classical Key Exchange

- Bootstrap problem: how do Alice, Bob begin?
  - Alice can’t send it to Bob in the clear!
- Assume trusted third party, Cathy
  - Alice and Cathy share secret key k_A
  - Bob and Cathy share secret key k_B
- Use this to exchange shared key k_s
- Ideas?



Simple Protocol

Alice $\xrightarrow{\{\text{ request for session key to Bob }\}\, k_A}$ Cathy

Alice $\xleftarrow{\{\, k_s \,\}\, k_A \,\|\, \{\, k_s \,\}\, k_B}$ Cathy

Alice $\xrightarrow{\{\, k_s \,\}\, k_B}$ Bob

Problems

- How does Bob know he is talking to Alice?
  - Replay attack: Eve records message from Alice to Bob, later replays it; Bob may think he's talking to Alice, but he isn't
  - Session key reuse: Eve replays message from Alice to Bob, so Bob re-uses session key
- Protocols must provide authentication and defense against replay

Needham-Schroeder

Alice → Cathy: Alice || Bob || r_1

Alice ← Cathy: { Alice || Bob || r_1 || k_s || { Alice || k_s } k_B } k_A

Alice → Bob: { Alice || k_s } k_B

Alice ← Bob: { r_2 } k_s

Alice → Bob: { $r_2 - 1$ } k_s

Denning-Sacco Modification

- Assumption: all keys are secret
- Question: suppose Eve can obtain session key. How does that affect protocol?
  - Assuming Eve knows k_s

$$\text{Eve} \xrightarrow{\{\,\text{Alice} \,\|\, k_s\,\}\,k_B} \text{Bob}$$

$$\text{Eve} \xleftarrow{\{\,r_2\,\}\,k_s} \text{Bob}$$

$$\text{Eve} \xrightarrow{\{\,r_2 - 1\,\}\,k_s} \text{Bob}$$

Needham-Schroeder with Denning-Sacco Modification

Alice → Cathy: Alice || Bob || r_1

Alice ← Cathy: { Alice || Bob || r_1 || k_s || { Alice || T || k_s } k_B } k_A

Alice → Bob: { Alice || T || k_s } k_B

Alice ← Bob: { r_2 } k_s

Alice → Bob: { $r_2 - 1$ } k_s

Kerberos

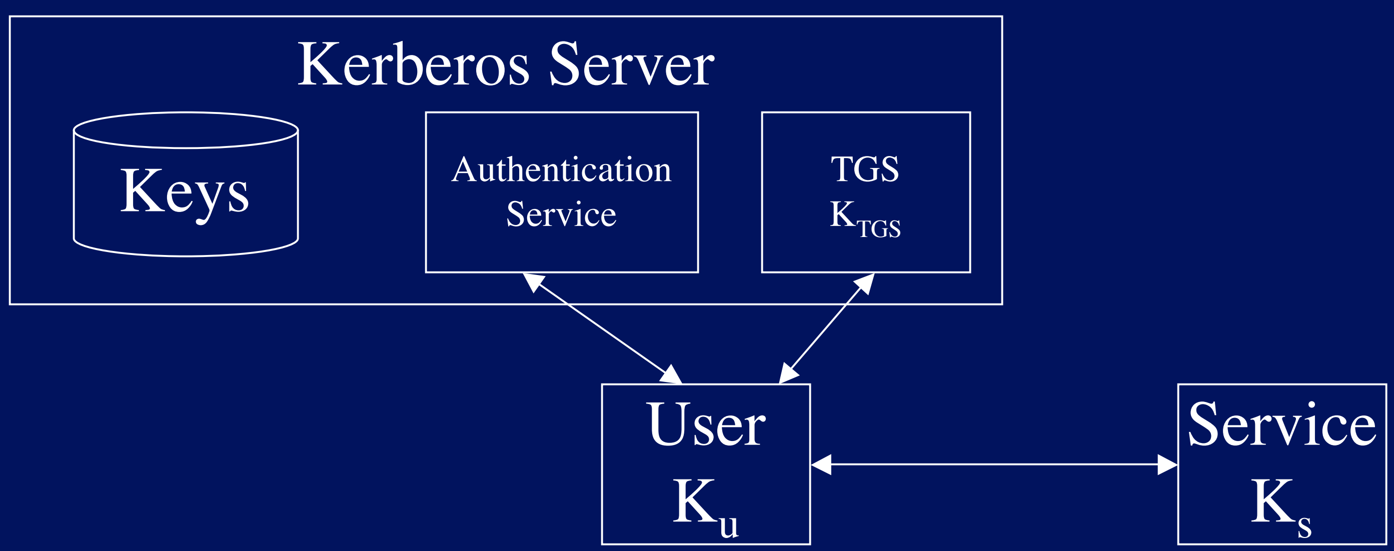
Copyright © 2004 Konstantin Beznosov

What is Kerberos?

- Authentication system
  - Based on Needham-Schroeder with Denning-Sacco modification
  - Central server plays role of trusted third party ("Cathy")
- Ticket
  - Issuer vouches for identity of requester of service
- Authenticator
  - Identifies sender

Idea

- User *u* authenticates to Kerberos server
  - Obtains ticket $T_{u,TGS}$ for ticket granting service (TGS)
- User *u* wants to use service *s*:
  - User sends authenticator A_u, ticket $T_{u,TGS}$ to TGS asking for ticket for service
  - TGS sends ticket $T_{u,s}$ to user
  - User sends A_u, $T_{u,s}$ to server as request to use *s*



Ticket

- Credential saying issuer has identified ticket requester
- Example ticket issued to user *u* for service *s*

$$T_{u,s} = s \mid\mid \{\ u \mid\mid u\text{'s address} \mid\mid \text{valid time} \mid\mid k_{u,s}\ \}\ k_s$$

where:

- $k_{u,s}$ is session key for user and service
- Valid time is interval for which ticket valid
- *u*'s address may be IP address or something else
  - Note: more fields, but not relevant here

Authenticator

- Credential containing identity of sender of ticket
  - Used to confirm sender is entity to which ticket was issued
- Example: authenticator user *u* generates for service *s*

$$A_{u,s} = \{ u \mid\mid \text{generation time} \mid\mid k_t \} k_{u,s}$$

where:

- k_t is alternate session key
- Generation time is when authenticator generated
  - Note: more fields, not relevant here

Protocol

user → *AS*: $user \parallel TGS$

user ← *AS*: $\{ k_{u,TGS} \} k_u \parallel T_{u,TGS}$

user → TGS: $service \parallel A_{u,TGS} \parallel T_{u,TGS}$

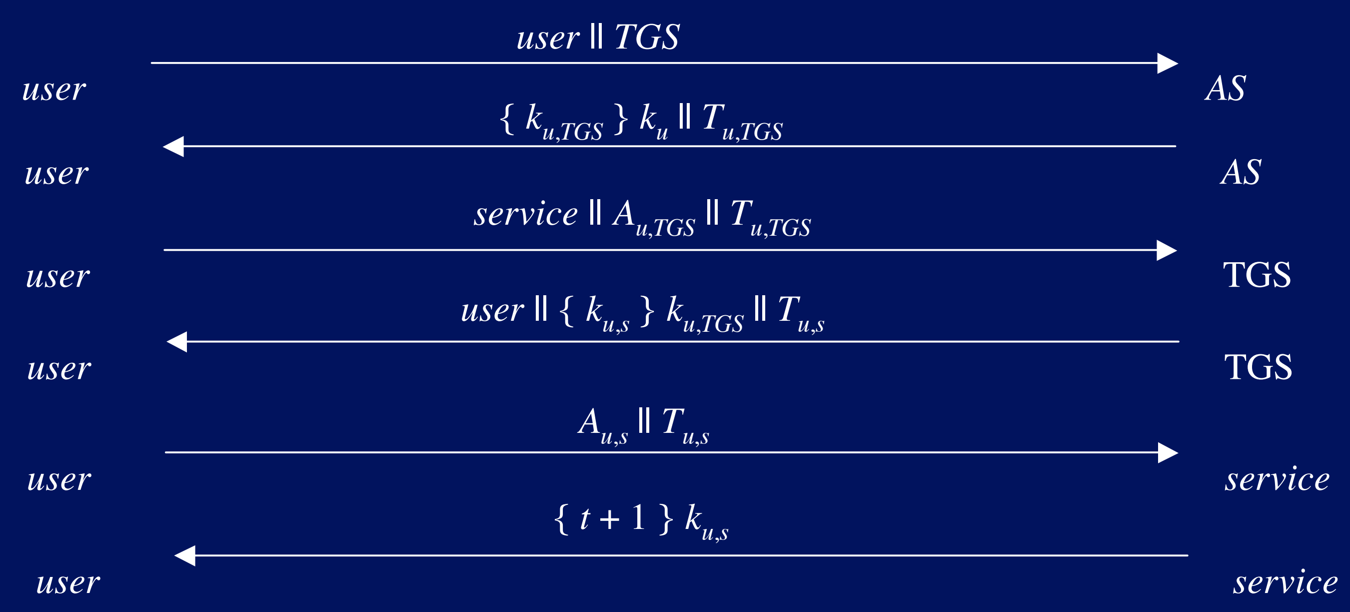
user ← TGS: $user \parallel \{ k_{u,s} \} k_{u,TGS} \parallel T_{u,s}$

user → *service*: $A_{u,s} \parallel T_{u,s}$

user ← *service*: $\{ t + 1 \} k_{u,s}$

Analysis

- First two steps
  - get user ticket to use TGS
  - User *u* can obtain session key only if *u* knows key shared with AS
- Next four steps
  - *u* gets and uses ticket for service *s*
  - Service *s* validates request by checking sender (using $A_{u,s}$)
  - Step 6 optional; used when *u* requests confirmation



Problems

- Relies on synchronized clocks
  - If not synchronized and old tickets, authenticators not cached, replay is possible
- Tickets have some fixed fields
  - Dictionary attacks possible
  - Kerberos 4 session keys weak (had much less than 56 bits of randomness)
    - researchers at Purdue found them from tickets in minutes

THE UNIVERSITY OF BRITISH COLUMBIA

Public Key Exchange

Copyright © 2004 Konstantin Beznosov

What is Public Key Key Exchange?

- Here interchange keys known
  - e_A, e_B Alice and Bob's public keys known to all
  - d_A, d_B Alice and Bob's private keys known only to owner
- Simple protocol
  - k_s is desired session key

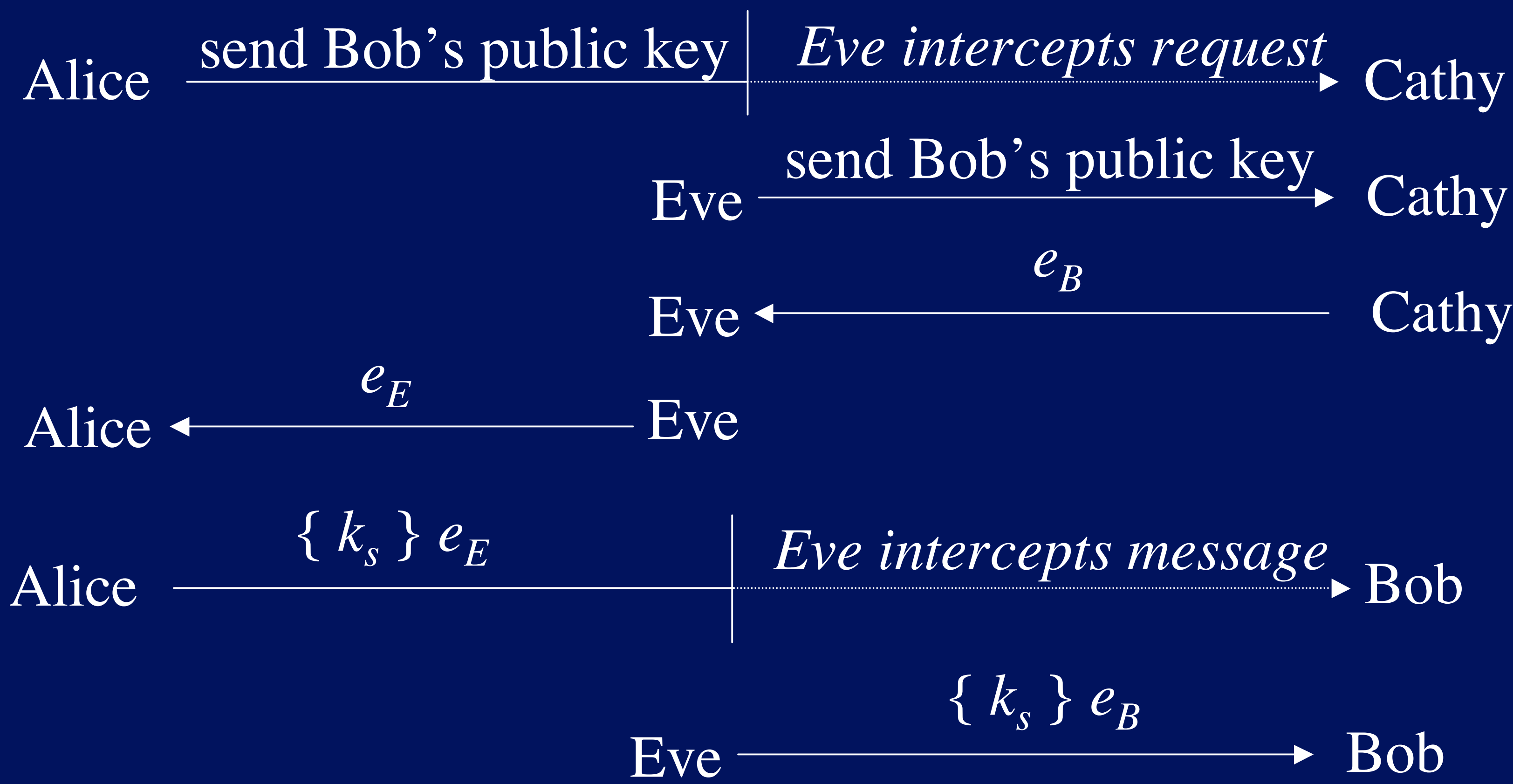
$$\text{Alice} \xrightarrow{\{\, k_s \,\}\, e_B} \text{Bob}$$

Problem and Solution

- Vulnerable to forgery or replay
  - Because e_B known to anyone, Bob has no assurance that Alice sent message
- Simple fix uses Alice's private key
  - k_s is desired session key

$$\text{Alice} \xrightarrow{\{\, \{\, k_s \,\}\, d_A \,\}\, e_B} \text{Bob}$$

Why Alice Can't Get Bob's Public Key



THE UNIVERSITY OF BRITISH COLUMBIA

Cryptographic Key Infrastructure

Copyright © 2004 Konstantin Beznosov

What's Cryptographic Key Infrastructure?

- Goal: bind identity to key
- Classical: not possible as all keys are shared
  - Use protocols to agree on a shared key (see earlier)
- Public key: bind identity to public key

Certificates

- Token (message) containing
  - Corresponding public key
  - Identity of principal (here, Alice)
  - Timestamp (when issued)
  - Other information (perhaps identity of signer)

signed by trusted authority (here, Cathy)

$$C_A = \{ e_A \mid\mid \text{Alice} \mid\mid T \} d_C$$

Use

- Cathy issues Alice's certificate
    - Creates certificate
    - Generates hash of certificate
    - Enciphers hash with her private key
- Bob gets Alice's certificate
  - Validates
    - Obtains issuer's public key
    - Deciphers enciphered hash
    - Recomputes hash from certificate and compare
- Problem?
  - Bob needs Cathy's public key to validate certificate
  - Two approaches: Merkle's trees, signature chains

Certificate Signature Chains

- Purpose: getting issuer's public key
- Solutions:
  - tree-like hierarchies
  - Webs of trust (PGP)

X.509 Chains

- Some certificate components in X.509v3:
  - Version
  - Serial number
  - Signature algorithm identifier: hash algorithm
  - Issuer's name; uniquely identifies issuer
  - Interval of validity
  - Subject's name; uniquely identifies subject
  - Subject's public key
  - Signature: enciphered hash

PGP Certification

- Single certificate may have multiple signatures
- Notion of "trust" embedded in each signature
  - Range from "untrusted" to "ultimate trust"
  - Signer defines meaning of trust level (no standards!)

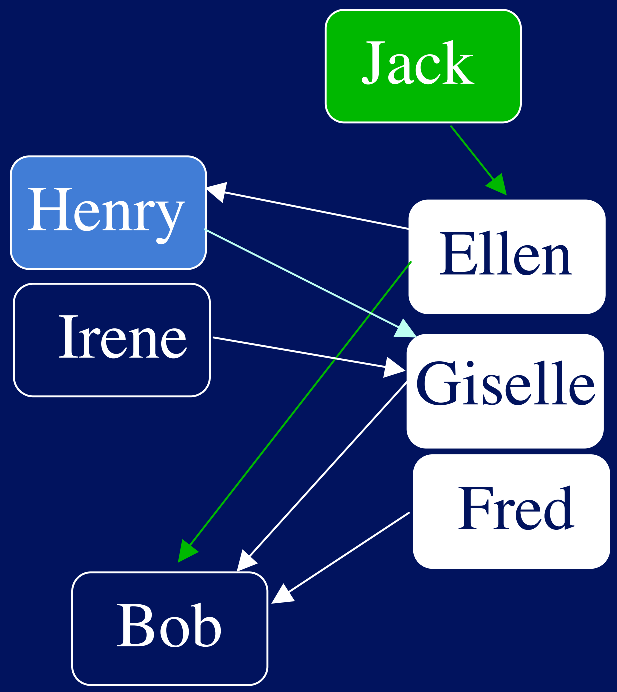
Validating Certificates

Alice needs to validate Bob's OpenPGP cert

- Does not know Fred, Giselle, or Ellen

1. Alice gets Giselle's cert
   - Knows Henry slightly, but his signature is at "casual" level of trust
2. Alice gets Ellen's cert
   - Knows Jack, so uses his cert to validate Ellen's, then hers to validate Bob's

Arrows show signatures
Self signatures not shown



Key Revocation

- Why revoke a key?
  - Certificates invalidated *before* expiration
    - Usually due to compromised key
    - May be due to change in circumstance (*e.g.,* someone leaving company)
- Problems
  - Entity revoking certificate authorized to do so
  - Revocation information circulates to everyone fast enough

CRLs

- *Certificate revocation list* lists certificates that are revoked
- X.509: only certificate issuer can revoke certificate
  - Added to CRL
- PGP: signers can revoke signatures; owners can revoke certificates, or allow others to do so

THE UNIVERSITY OF BRITISH COLUMBIA

Quantum Key Distribution (QKD)

Slides from this section are adopted from
Ravi Kumar Balachandran's slides on QKD available at
http://cse.unl.edu/~ashok/CSCE990Seminar/slides/ravib.ppt

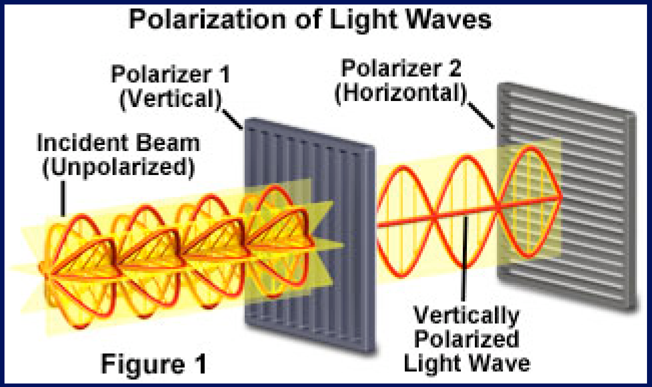
Copyright © 2004 Konstantin Beznosov

Why QKD?

- The security of all current encryption algorithms depend on solving some computationally difficult problems
  - RSA – factoring large prime numbers
  - Symmetric ciphers -- brute force search of the key.
- Quantum computers (in the future) can speed up this process making such ciphers trivial to break
- Quantum theory also forms the basis for QKD

Polarization of light

- Every photon from a light source vibrates in all directions – unpolarized light
- When light is passed through a polarizer, the out coming light is said to be polarized with respect to the polarizer



QKD Scheme

Alice's Sending Bases	×	+	+	×	×	+	×	+
Alice's Values	↘	↔	↔	↗	↘	↔	↗	↕
Bob's Receiving Bases	+	×	+	×	+	+	×	×
Bob's Values	↔	↗	↔	↗	↔	↔	↗	↘
Alice Confirms			✓	✓		✓	✓	
Key			1	0		1	0	

Implementation of QKD

- First prototype in 1989, two computers separated by a distance of 32 cm by Bennet
- Los Alamos – 1996 – 14 Km with fiber in the field
- British Telecom – 1998 – 30 Km
- Successful tests have been done over distances of 1.6 Km with no waveguide
- March 2002 – 67 Km using optical fiber working at 1550nm
- October 2003 -- world's first quantum cryptographic network: 6 QKD nodes in Cambridge, MA; 22 Km
- High-grade key material at rate 5Kb/s