

# OUTLINE

- **Approaches to Message Authentication**
- **Public-Key Cryptography Principles**
- **Public-Key Cryptography Algorithms**
- **Key Management**
- **PGP Practical Example**



# Authentication

- **Requirements - must be able to verify that:**
  1. **Message came from apparent source or author,**
  2. **Contents have not been altered,**
  3. **Sometimes, it was sent at a certain time or sequence.**
- **Protection against active attack (falsification of data and transactions)**



# Approaches to Message Authentication

- **Authentication Using Conventional Encryption**
  - Only the sender and receiver should share a key
- **Message Authentication without Message Encryption**
  - An authentication tag is generated and appended to each message
- **Message Authentication Code**
  - Calculate the MAC as a function of the message and the key.  $MAC = F(K, M)$



# Cryptographic Hash

- Producing *hash values* for accessing data or for security.
- A hash value (or simply *hash*) is a number generated from a string of text.
  - The hash is substantially smaller than the text itself, and is generated by a formula in such a way that it is extremely unlikely that some other text will produce the same hash value.
- Hashes play a role in security systems where they're used to ensure that transmitted messages have not been tampered with.
  - The sender generates a hash of the message, encrypts it, and sends it with the message itself.
  - The recipient then decrypts both the message and the hash, produces another hash from the received message, and compares the two hashes.
  - If they're the same, there is a very high probability that the message was transmitted intact.

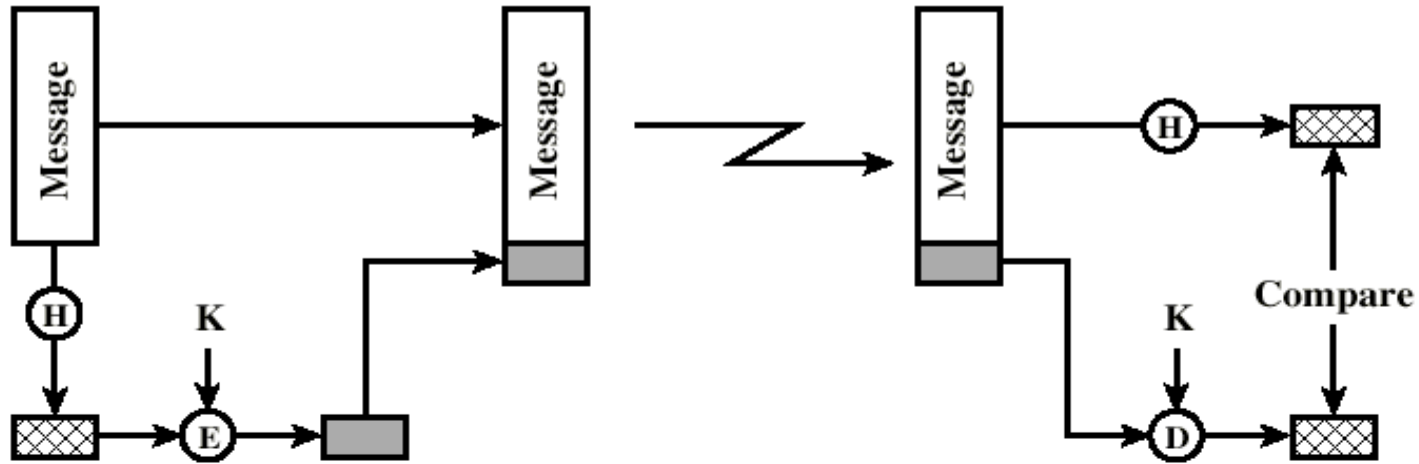


# Trivial Hashing Example

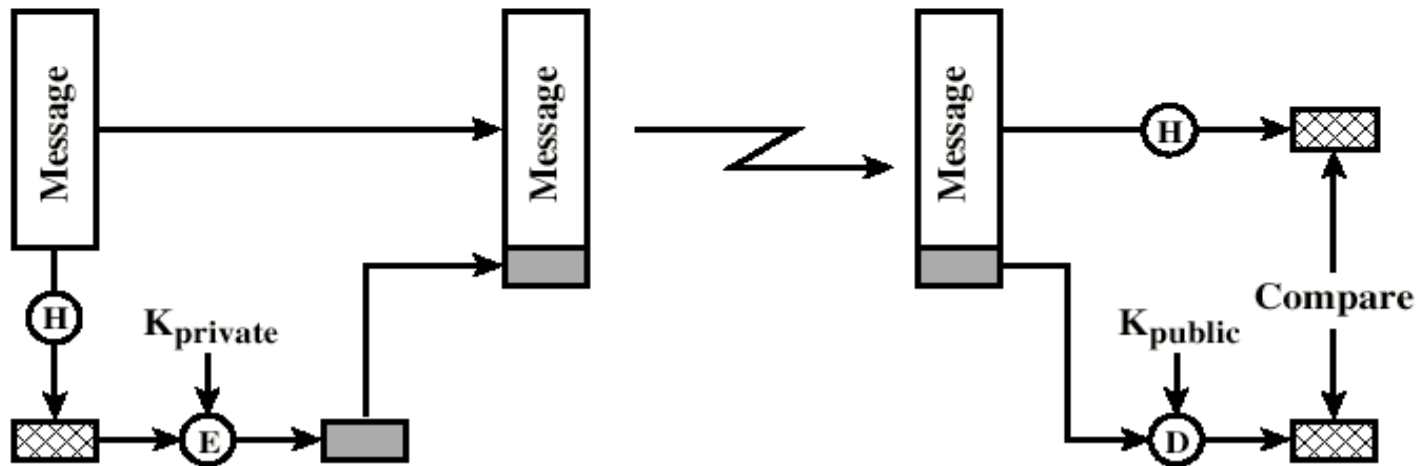
- Hashing is also a common method of accessing data records. Consider, for example, a list of names:
  - John Smith
  - Sarah Jones
  - Roger Adams
- To create an index, called a *hash table*, for these records, you would apply a formula to each name to produce a unique numeric value. So you might get something like:
  - 1345873 John Smith
  - 3097905 Sarah Jones
  - 4060964 Roger Adams
- Then to search for the record containing *Sarah Jones*, you just need to reapply the formula, which directly yields the index key to the record.
- This is much more efficient than searching through all the records till the matching record is found.



# One-way HASH function



(a) Using conventional encryption

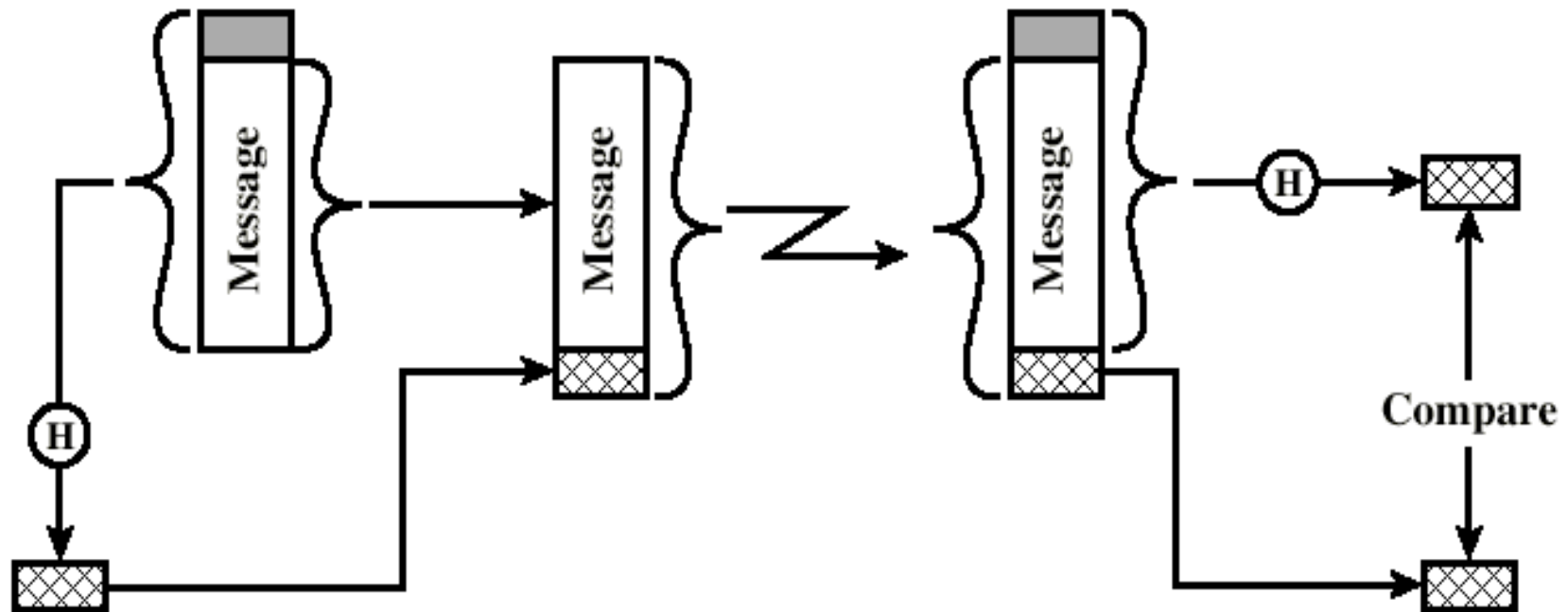


(b) Using public-key encryption



# One-way HASH function

- Secret value is added before the hash and removed before transmission.



(c) Using secret value



# Secure HASH Functions

- Purpose of the HASH function is to produce a “fingerprint.”
- Properties of a HASH function  $H$  :
  1.  $H$  can be applied to a block of data at any size
  2.  $H$  produces a fixed length output
  3.  $H(x)$  is easy to compute for any given  $x$ .
  4. For any given block  $x$ , it is computationally infeasible to find  $x$  such that  $H(x) = h$
  5. For any given block  $x$ , it is computationally infeasible to find  $y \neq x$  with  $H(y) = H(x)$ .
  6. It is computationally infeasible to find any pair  $(x, y)$  such that  $H(x) = H(y)$



# Public-Key Cryptography Principles

- The use of two keys has consequences in: key distribution, confidentiality and authentication.
- The scheme has six ingredients
  - Plaintext
  - Encryption algorithm
  - Public and private key
  - Ciphertext
  - Decryption algorithm



# Secure Email Research: Verification and Validation of PGP

**Lt.Col. John Hill, USA, US Military Academy**

**Major Curt Carver, USA, US Military Academy**

**Capt. Jeff Humphries, USAF, US Air Force Academy**

**Capt. David Rosen, USAF, National Security Agency**

**Lt.Col. Drew Hamilton, USA, SPAWAR**

**CDR. Mari Obninski, USN, US Pacific Command**



# Assumptions

- Requirement exists for secure email to transmit sensitive but unclassified information for joint and combined operations in the Pacific Theater.
- Commercial Internet preferred method of transferring messages.
- No special encryption hardware should be required.
- Cost is a factor.
- Source code required to evaluate encryption evaluation.

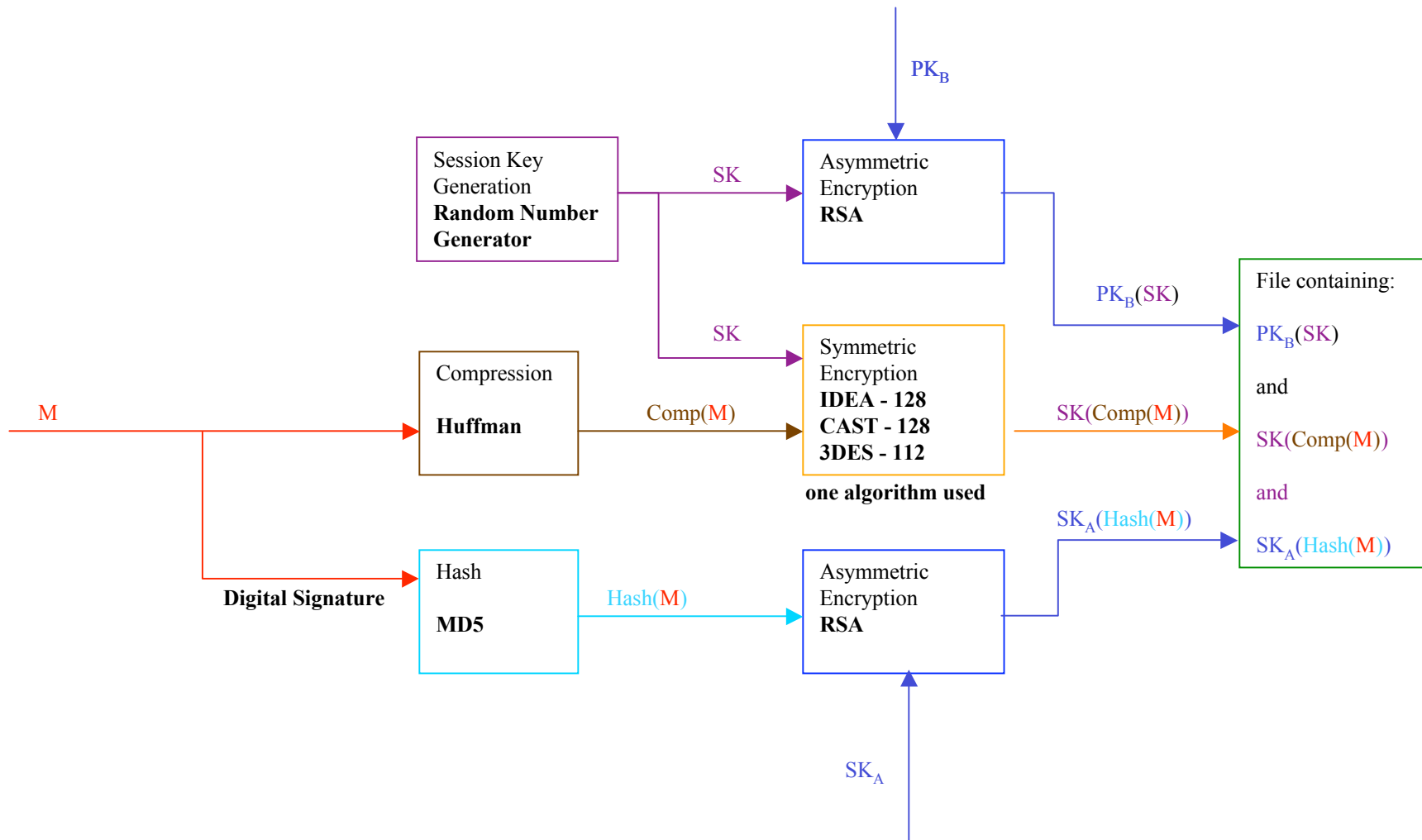


# PGP

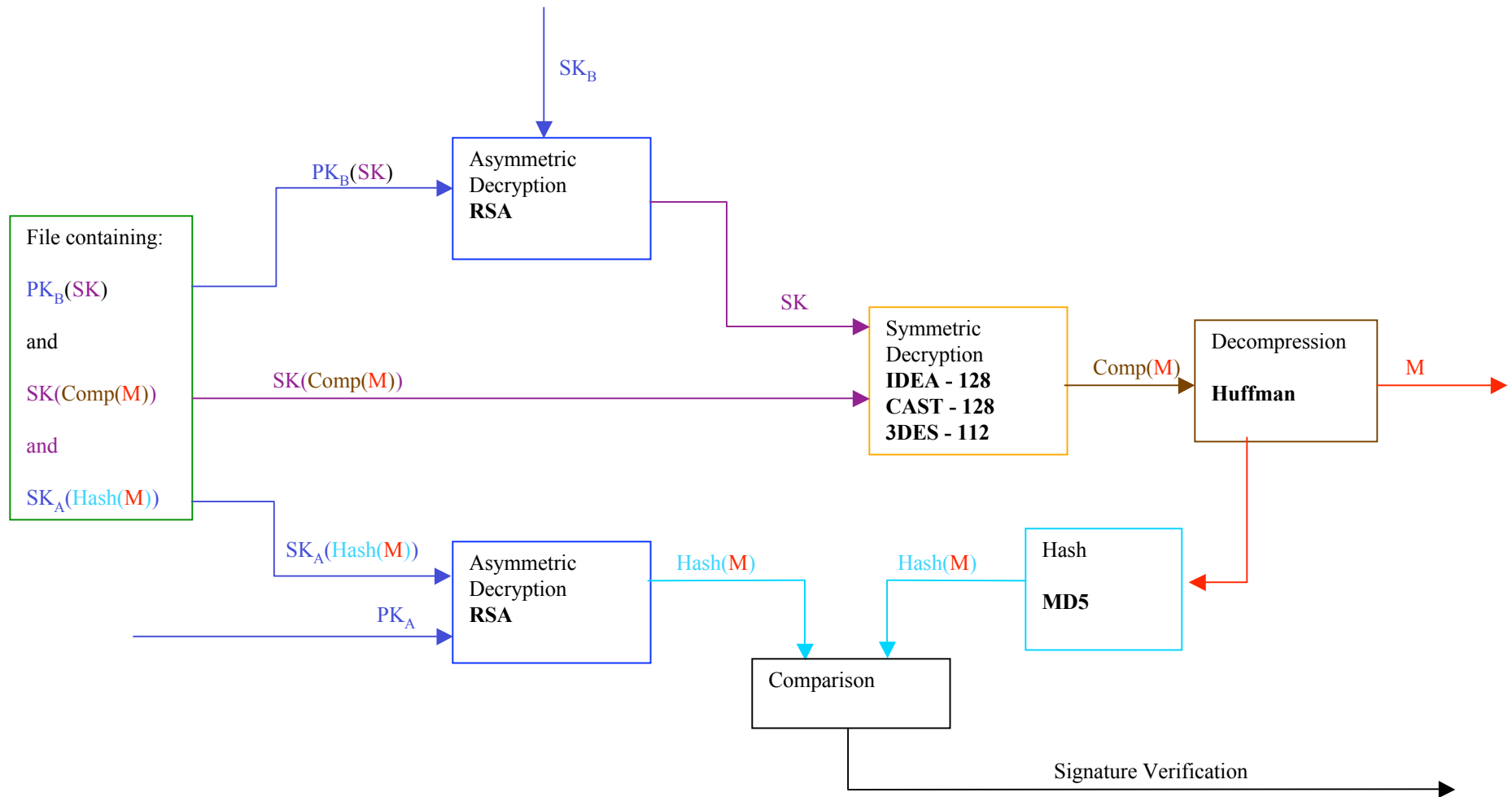
- **Freeware available from MIT (version 6) and Network Associates (version 6).**
- **Source code publicly available.**
- **PGP versions have been publicly available and tested since 1991.**
- **However, as with most reported results, insufficient information is provided to replicate the tests.**



# PGP Sender



# PGP Receiver



# Endpoint Concerns

- **Physical Security**
- **Tempest**
- **Compromise (pass phrase, Session Key<sub>a</sub>, secret keys)**
- **Public Key Tampering**
- **Viruses or Trojan Horses (PGP)**
- **Social Engineering**
- **Audio/Video Surveillance**
- **Key Management**



# General Session Key Issues

- **Compromise of Keys (Stolen Keys)**
- **Destroying keys**
- **Seed Attacks**
  - Seed stored on hard drive
  - Not truly random
    - short period
    - deterministic
  - Tested version of PGP:
    - changes seed for each session
    - new seed created from old seed and monitored keyboard activity



# Encryption Attacks

- **Symmetric Encryption**
  - **Brute force**
  - **Flaw in implementation**
- **Asymmetric Encryption**
  - **Brute force**
  - **Factoring (RSA)**
  - **Flaw in implementation**



# Testing by JFPO Team

- **Review source code to verify correct algorithm implementation**
  - Encryption algorithms extracted from source code and desk checked.
- **Brute force on 112-bit 3DES Symmetric**
  - Software attack only, theoretical evaluation of DES-breaking machines
- **Brute force on 768-bit RSA Asymmetric**
  - Search for all primes in the key space
- **Factoring on RSA 768-bit**
  - Concur with LTC Gibson's recommendation for 2048 bit default
  - 768-bit is the shortest key allowable in PGP version 6.02



# Agenda

- **Introduction and Problem Statement**
- **Research Methodology**
- **Attack Methodologies**
  - Brute Force Keyspace Search
  - Prime Number Sieve
  - Endpoint Management
- **Conclusion**



# Problem Statement

- The PACOM J6 is interested in an unclassified risk assessment on the use of commercial or publicly available encryption systems to transmit sensitive (but not classified) information over an unsecure network.
- The investigators use one particular system, Pretty Good Privacy (PGP), to demonstrate cryptographic algorithms and systems by examining its internal operation in detail as well as potential weaknesses.



# Research Methodology

- **Examine RSA, Triple-DES and IDEA algorithms and endpoint management weaknesses in PGP.**
- **Conduct brute force attacks using single and multiple computers against the symmetric and asymmetric encryption algorithms.**
- **Extrapolate the results to provide a risk assessment.**



# Attack Methodology

- **Brute Force Keyspace Search:** Compromise sensitive email traffic by attempting to decrypt the message using every possible key.
- **Prime Number Sieve:** Compromise the public key component of the encryption algorithm by determining the underlying prime numbers used in the encryption algorithm.
- **Endpoint Management:** Use social engineering or other techniques such as dumpster diving to determine the encryption key.



# Brute Force Keyspace Search

Compromise sensitive email traffic by attempting to decrypt the message using every possible key.



# Effects on Search Time

## (Number of Bits)

- Each time the key length is increased by one bit the size of the key space doubles which doubles the search time
  - When the key has 8 bits, the size of the search space is  $2^8 = 256$  possible keys
  - When the key has 9 bits, the size of the search space is  $2^9 = 512$  possible keys
- In the actual test runs, the search times follow this  $2^{\text{NumBits}}$  rule: for each 1-bit increase in key size the search time doubles



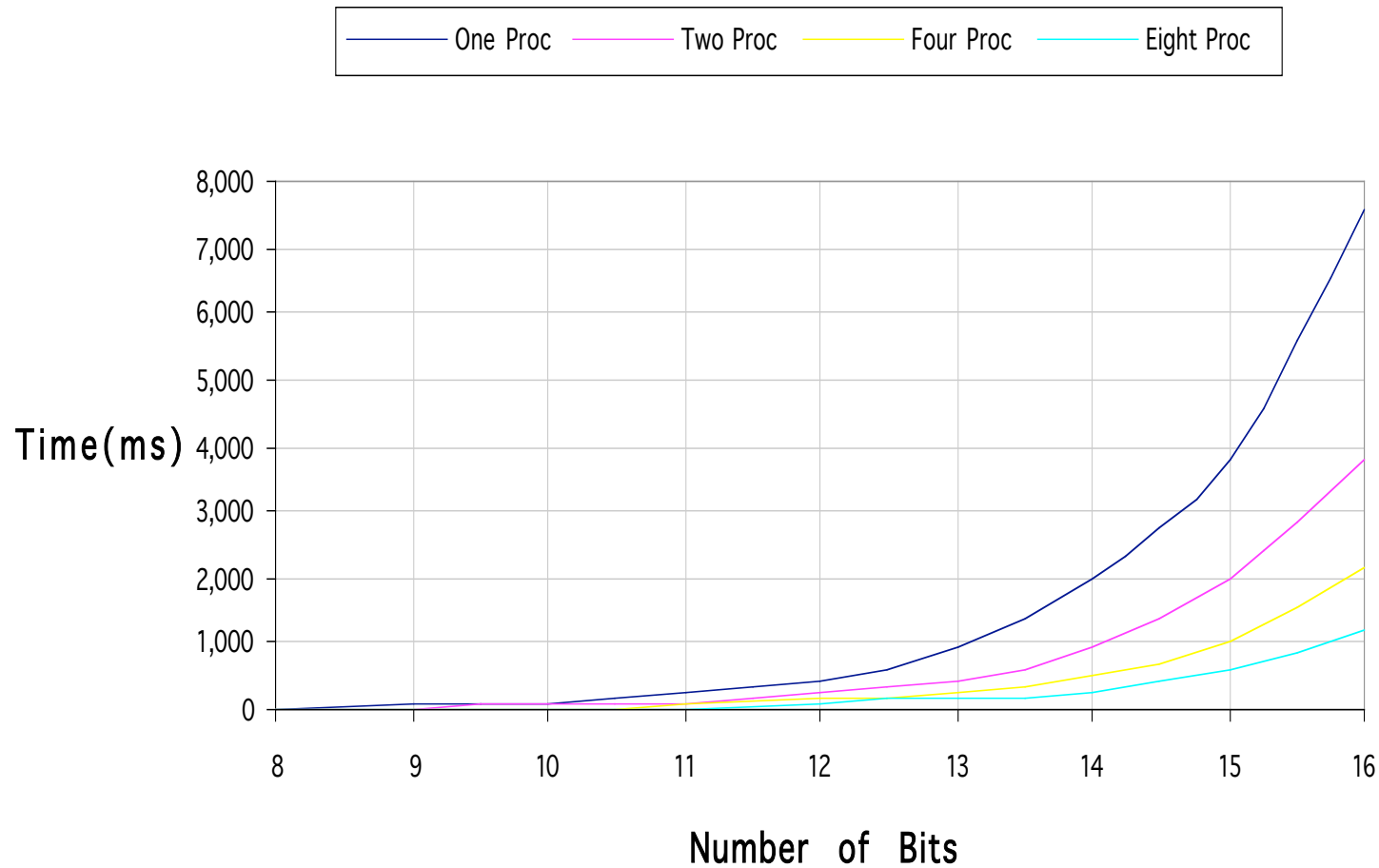
# Effects on Search Time

## (Number of Systems)

- Each time the number of systems is doubled the search time is halved
  - When the key has 8 bits and the number of systems is 16, it will take  $2^8/16 = 16$  units of time.
  - When the key has 8 bits and the number of systems is 32, it will take  $2^8/32 = 8$  units of time.
- In the actual test runs the search times follow this  $\text{SearchTime}/\text{NumSystems}$  rule: for each doubling of the number of systems the search time is halved



# Effects on Search Time



# Results of Test Runs

## (Single Machines)

- Bit-length required to secure a message against brute force search by one of the following machines:

	Sparc5	Celeron 300a	Sparc10
<b>1 month</b>	<b>34</b>	<b>38</b>	<b>38</b>
<b>1 year</b>	<b>38</b>	<b>41</b>	<b>42</b>
<b>10 years</b>	<b>42</b>	<b>44</b>	<b>45</b>
<b>100 years</b>	<b>45</b>	<b>48</b>	<b>48</b>
<b>1,000 years</b>	<b>48</b>	<b>51</b>	<b>52</b>
<b>1,000,000 years</b>	<b>58</b>	<b>61</b>	<b>62</b>



## Results of Test Runs (Multiple Machines)

- **Bit-length required to secure a message against distributed brute force search by Sparc5 machines:**

	<b>1</b>	<b>2</b>	<b>4</b>	<b>8</b>
<b>1 month</b>	<b>34</b>	<b>35</b>	<b>36</b>	<b>37</b>
<b>1 year</b>	<b>38</b>	<b>39</b>	<b>40</b>	<b>41</b>
<b>10 years</b>	<b>42</b>	<b>43</b>	<b>44</b>	<b>45</b>
<b>100 years</b>	<b>45</b>	<b>46</b>	<b>47</b>	<b>48</b>
<b>1,000 years</b>	<b>48</b>	<b>49</b>	<b>50</b>	<b>51</b>
<b>1,000,000 years</b>	<b>58</b>	<b>59</b>	<b>60</b>	<b>61</b>



# Prime Number Sieve

Compromise the public key component of the encryption algorithm by determining the underlying prime numbers used in the encryption algorithm.



# Using Prime Numbers

- Algorithms ‘sieve’ through possible prime number factors, then use linear algebra to narrow down guesses.
- Algorithms have evolved significantly over last several years.
- Best (current) technique is “General Number Field Sieve” (GNFS).



# RSA Prime Number Attacks

- GNFS is highly parallelizable over networks of workstations and recent attacks use massive resources around the Internet.
- RSA129 took 8 calendar months in 1994.
- RSA130 needed a few teraop-hours, equating to several months in 1995.
- RSA155 (512 bits) needs two teraop-months! The cracking of RSA155 has not been publicly announced.



# Endpoint Management

- **Compromised Pass Phrases & Secret Keys**
  - **Public Key Tampering**
    - **Deleted Files**
    - **Random Numbers**
    - **Attacks on IDEA**
  - **MD-5 Digital Signatures**



# Compromised Pass Phrases and Secret Keys

- **Vulnerability:** The simplest attack is if someone leaves the pass phrase for their secret key written down somewhere or stored in a file on the system. Users must keep backup copies of their secret key ring because there is only one copy of the secret key on the file system. Losing it will render useless all the copies of the user's public key that have spread throughout the world.
- **System Weakness:** If a hacker has the pass phrase and access to the encrypted private key, they can pose as the user



# Public Key Tampering

- **Vulnerability:** When the user uses someone's public key, they must make certain it has not been tampered with. A new public key from someone else should be trusted only if the user got it directly from its owner, or if it has been signed by someone the user trusts.
- **System Weakness:** By substituting a public key, a hacker can impersonate a user in all future transactions.



## Deleted Files

- **Vulnerability:** When a file is encrypted and then the original plaintext file is deleted, the operating system doesn't actually physically erase the data.
- **System Weakness:** If an attacker reads these deleted disk blocks after they have been deallocated, he could recover the plaintext.



# Random Numbers

- **Vulnerability:** PGP uses a cryptographically strong pseudorandom number generator for creating temporary conventional session keys. The seed file for this is named *randseed.bin*. This file is randomized after every use.
- **System Weakness:** There exists the theoretical possibility that if an attacker has access to this file, they may be able to use a side channel crypto-analysis attack to break the PGP encryption.



# IDEA

- **Vulnerability** : PGP depends on the IDEA (International Data Encryption Algorithm) block cipher. If a weakness can be found in IDEA encryption, it can be exploited to compromise PGP.
- **System Weakness**: IDEA was developed by a distinguished team of cryptographic experts and has had extensive security analysis and peer review from some of the best crypto-analysts in the unclassified world. Given that breaking a 128 bit IDEA key would be equivalent to cracking a 3100-bit RSA key, this algorithm currently appears secure.



# MD5 – Digital Signatures

- **Vulnerability:** To create a digital signature, PGP encrypts with the user's secret key. However, PGP doesn't actually encrypt the entire message with the secret key because of the time overhead. Instead, PGP encrypts a "message digest" which is a hash of the message.
- **System Weakness:** When two messages hash to the same value it is called a collision and the birthday attack can attempt to exploit this phenomena. The birthday attack is a statistical probability problem. Given  $n$  inputs and  $k$  possible outputs, (MD5 being the function to take  $n \rightarrow k$ ) there are  $n(n-1)/2$  pairs of inputs. For each pair, there is a probability of  $1/k$  of both inputs producing the same output. So, if you take  $k/2$  pairs, the probability will be 50% that a matching pair will be found. If  $n > \sqrt{k}$ , there is a good chance of finding a collision. In MD5's case,  $2^{64}$  messages need to be tried. This is not a feasible attack given today's technology.



# Conclusions

- **We have independently verified components of PGP. This verification is applicable to other security systems.**
- **The principal threat to PGP is non-cryptographic.**
- **There is overhead to implementing PGP in a organization. Managing 20 systems is feasible; managing 100 is difficult; in-between depends on your level of pain.**



# Supporting Slides



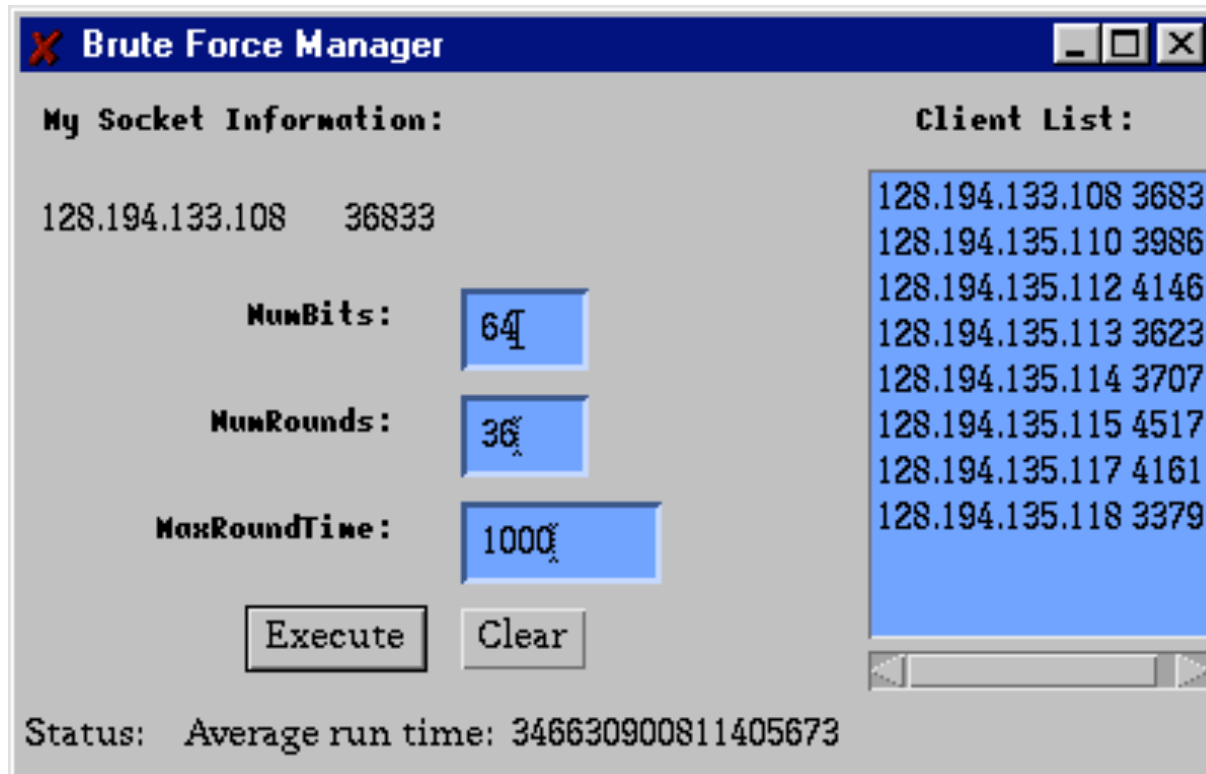
# Brute Force Manager

- **Accepts connections and disconnections from Brute Force Runners**
- **Accepts instructions for**
  - the number of bits in the key
  - the number of rounds (for statistical validity)
  - the maximum time (in ms) to spend on a round
- **Divides the search space ( $2^{\text{NumBits}}$ ) between the Brute Force Runners**
- **Collects the results from the Brute Force Runners and determines the extrapolated average run time (in ms) for the entire search**



# Brute Force Manager (Image)

## Interface



# Brute Force Runner

- **Accepts the Address and the Port for the Brute Force Manager's socket**
- **Connects and disconnects from the Brute Force Manager**
- **Receives instructions from the Brute Force manager that identify**
  - the key space to search
  - how long to search before extrapolating the results
  - the number of times to perform the search
- **Reports the results to the Brute Force Manager**



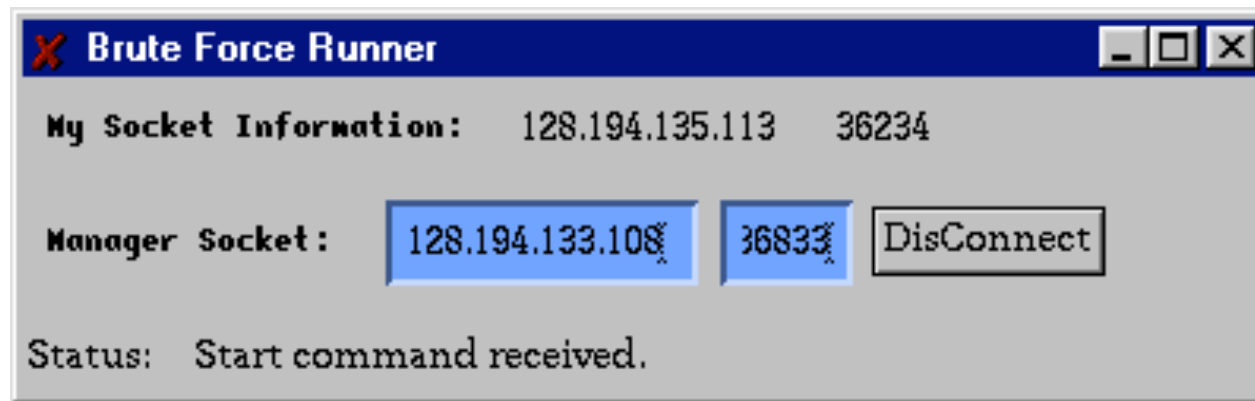
# Brute Force Runner

- **Accepts the Address and the Port for the Brute Force Manager's socket**
- **Connects and disconnects from the Brute Force Manager**
- **Receives instructions from the Brute Force manager that identify**
  - the key space to search
  - how long to search before extrapolating the results
  - the number of times to perform the search
- **Reports the results to the Brute Force Manager**



# Brute Force Runner (Image)

## Interface



# Single Machine Brute Force Session

- **A session consists of**
  - the brute force search program running on a computer
  - input from the user about the bit-length of the key
  - input from the user about how many rounds to perform the search (helps statistical validity)
  - input from the user about how long to let a search run before extrapolating the total time
- **In the example sessions**
  - The program was run on a Sparc5, a Celeron 300a, and a Sparc10
  - Bit-lengths up to 512 were examined
  - There were 36 rounds used for each bit-length
  - The maximum round time was 1000ms



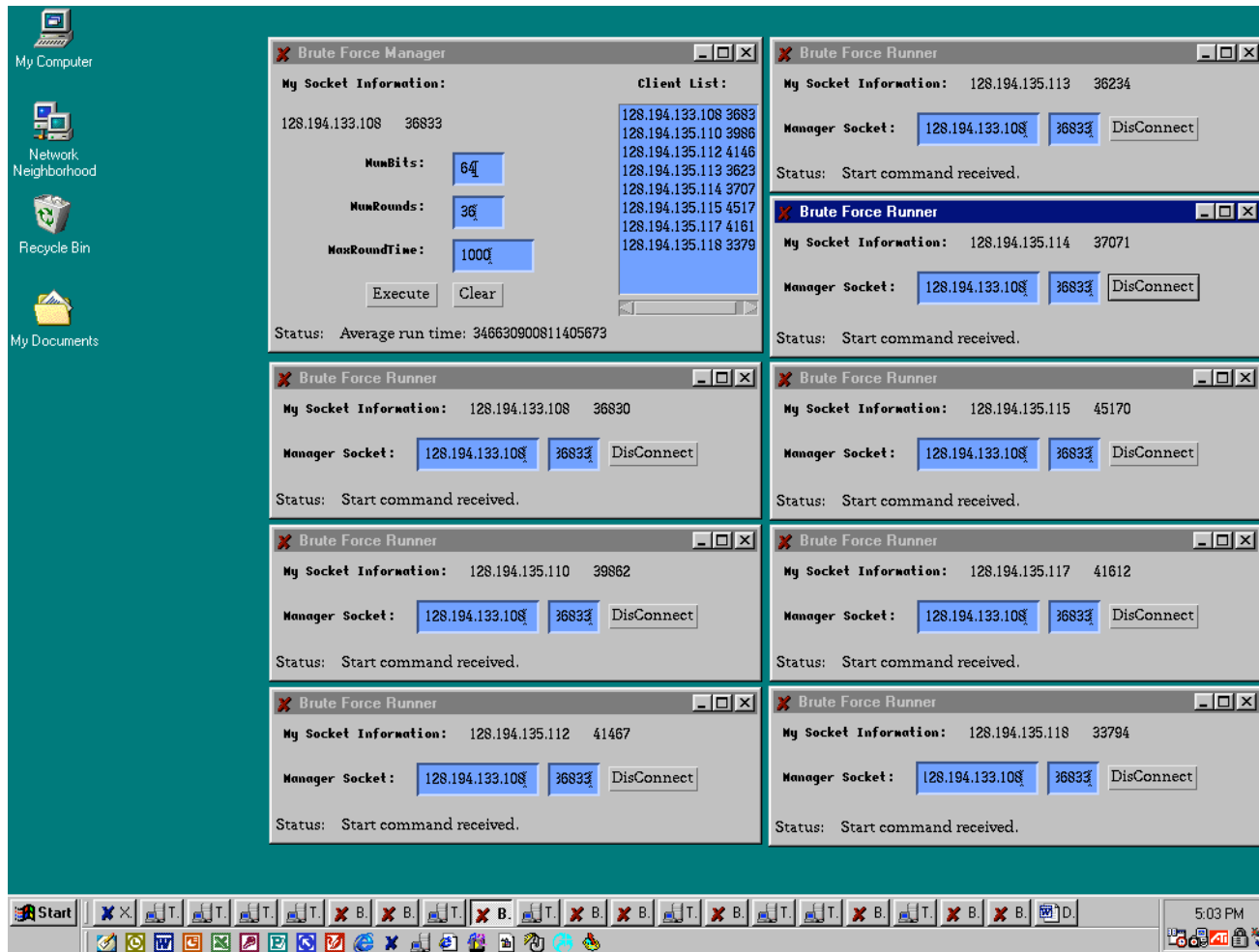
# Distributed Brute Force Session

- **A session consists of**
  - one instance of the Brute Force Manager
  - as many instances of the Brute Force Runner as desired
  - instructions from the user to perform searches
- **In the example session**
  - The Brute Force Manager ran on a Sun Sparc10 machine running Unix (Solaris)
  - The Brute Force Runners all ran on separate Sun Sparc5 machines running Unix (Solaris)
  - The controlling machine was a Celeron 300a running Windows98
    - The screen captures come from this machine
  - All communications are through TCP



# Distributed Brute Force Session (Image)

Screen capture of session with 8 runners



# Scenario Based on Actual Results

- The protagonist has a secret that needs to be kept for 50 years and wants to protect it with encryption. Therefore, he needs a key length that requires 100 years to search (the average discovery time would be 50 years).
- The antagonist has a Sparc5 available to search for the key
- The protagonist looks at the results of the test runs for one Sparc5 machine and determines that he must use a 45-bit key to protect the secret.
- The antagonist states that he will purchase another Sparc5 and have the secret in 25 years.
- The protagonist adds one bit to the key and again has 50 years of security



# Scenario (continued)

- The antagonist, in disgust, says “Fine, I’ll buy two more machines (a total of four).”
- Protagonist: “I’ll add one more bit and make it a 47-bit key. It will still take you 50 years.”
- Antagonist: “I’ll buy four more machines (total of eight) and have it in 25 years.”
- --- later ---
- Protagonist: “I’ll make it a 64-bit key.”
- Antagonist: “I’ll buy 262,144 Sparc5 machines and have it in 25 years.”
- Clearly, it is much easier and less expensive for the protagonist

