Zero-Interaction Authentication

Mark Corner, Brian Noble
University of Michigan

Presented by
Martin Meyer

Overview

- ➢ Introduction
- Design of System
  - – Encryption Model
  - – Authentication Token
  - – File Ownership
  - – Departure and Return
- Implementation
- Evaluation

What is ZIA?

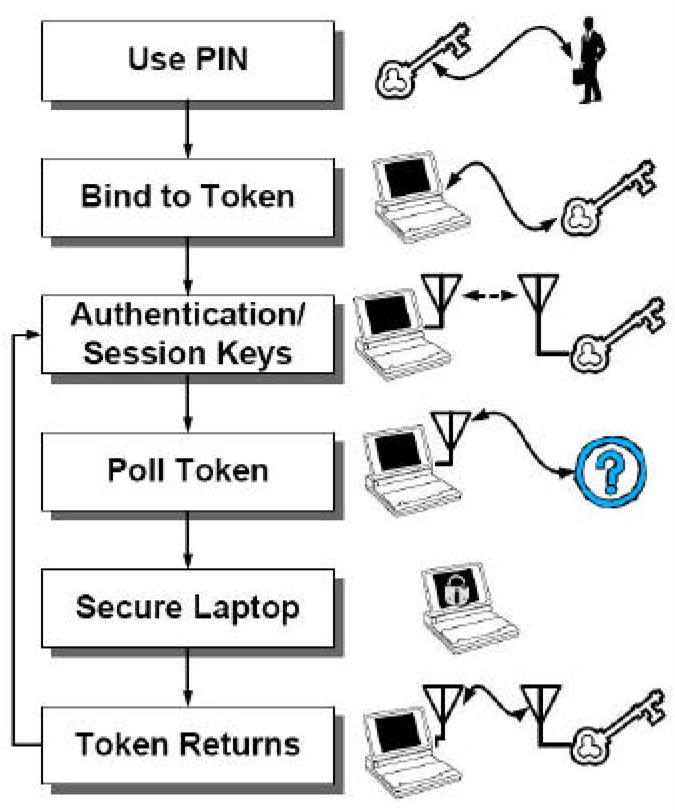
- Any small device can be stolen
- If an unprotected laptop is stolen, assume all data is compromised and invalidate it
- Encryption of data required authentication
- If security inconveniences users they will try to bypass it
- If data is secure, just buy a replacement and restore from backups
- Create a way to automatically authenticate the user when they are nearby

Securing with Encryption

- Encryption most common defense
  - Requires initial decryption key
  - Key remains for duration of session
  - Steal lappy while logged in and data is compromised
  - To be secure, frequent reauthorization
  - Reauthorization is burdensome for user

Authentication Token

- User can wear an *authentication token*
  - Small, unobtrusive device
  - Wireless authentication with computer when in range
  - Token is less vulnerable to theft since not often set down



This figure shows the process for authenticating and interacting with the token. Once an unlocked token is bound to a laptop, ZIA negotiates session keys and can detect the departure of the token.

Figure 2: Token Authentication System

Overview

- Introduction
- ➢ Design of System
  - – Encryption Model
  - – Authentication Token
  - – File Ownership
  - – Departure and Return
- Implementation
- Evaluation

Design Outline

- Provide encryptions without affecting performance or usability
- Since token should be low-performance, encrypt at host
- Session between token and laptop encrypted to secure key transfers
- Prevent successful hacks

Threat and Trust Model

- Focus on defense against physical possession of laptop
  - Leaving a session opened is bad
  - Console access can result in root access
  - Can bypass OS entirely
- Protect from exploitation of wireless link
  - Observation, modification, insertion of messages
  - Eavesdropping to acquire key information

Threat and Trust (cont’d)

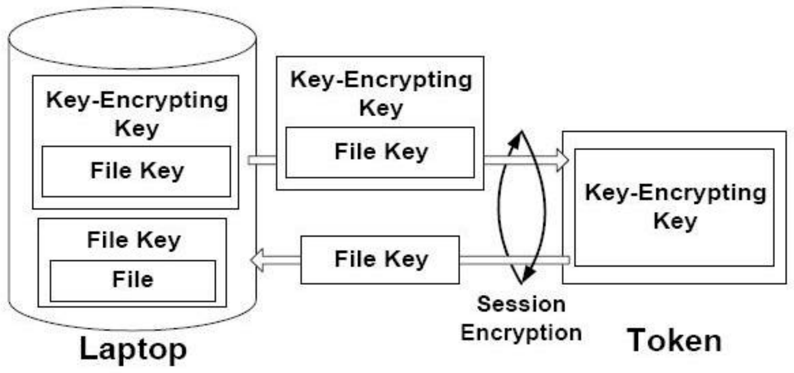
- Assume central administrative domain
- Assume a central authority for key management and rights revocation
- The token and laptop form a trusted computing base

Trust Model Shortcomings

- Does NOT defend against trusted but malicious users
- No protection for remote exploits
- Jamming of wireless channel can deny access to a user's files

Key-Encrypting Keys – $K_k(K_e)$

- Don't want to save every single key on the token
- Encrypt the files with one key, but encrypt that key with another key
  - The encryption key is K_e
  - The key-encrypting key is $K_k(K_e)$
- If user rights change, just re-encrypt all K_e with a new K_k



This figure illustrates the process of file key acquisition. Encrypted file keys are read from disk and shipped to the token. The token decrypts it and returns the file key. Traffic between the laptop and token is encrypted, preventing eavesdroppers from obtaining file keys.

Figure 1: Decrypting File Encrypting Keys

Key-Encrypting Keys (cont'd)

- Local administrative authority responsible for K_k assignments
- Each laptop needs an asymmetric key for establishing session encryption
- Key-encrypting keys must remain in escrow in case token is lost
  - Does not need to be highly available

Users, Groups, Doom.

- **Laptops usually assigned to specific users, but sometimes more than one**
- **Allow the file system driver to implement this as it sees fit**
- **Can implement a user/group/world system similar to Unix**
  - **Can provide good user functionality**

Users, Goups, Doom. (cont'd)

- Design the K_e so it can be unencrypted by multiple keys
  - Access rights to files depend on which key used
  - $K_u(K_e)$ is a key for an owning user
  - $K_g(K_e)$ is a key for an owning group
  - $K_w(K_e)$ is a key for any authorized laptop user, one K_w per machine
  - Each K_e can be encrypted by a K_u & K_g

User/Group Management

- What happens when a user leaves a group?
  - Must change K_g to a new K_g'
- User may have access to previous K_e
  - need to create K_e' and re-encrypt all files
- Re-encrypting can be done incrementally
  - Distribute new K_g' to authorized users and update all K_e to accept K_g'
  - Incrementally re-encrypt files so they no longer accept K_g

Laptop Vulnerabilities

- When lappy is stolen, files are secure
- File keys and session keys are zeroed in memory
- Private key must remain on laptop
  - If attacker recovers private key he can impersonate lappy
- What if lappy is modified and returned?
  - Secure booting can protect from this

Token Vulnerabilities

- Tokens secure because they are worn, not carried
- Lost or stolen tokens could divulge their K_k
  - PIN-protected, tamper-resistant hardware can be used
- Since token is more secure than laptop, authentication can be less frequent
  - Once a day may be adequate to provide protection
  - Important to prevent incentive in stealing a token, then later the associated laptop

Token Vulnerabilities (cont'd)

- What if a laptop were stolen then the thief tailgated near an authorized user from same domain?
  - Thief could force stolen laptop to generate key decryption requests
- To prevent tailgating, enforce laptop/token *binding*
  - Similar to Bluetooth pairing
  - Binding ends with token's session
  - Tokens can be bound to more than one laptop
  - Laptops can be bound to more than one token

Token Authentication

- Goals for binding:
  - Mutual authentication
  - Transfer of session encryption key
- Hosts must be assigned a trusted public key
  - Use the same CA that assigns key-encrypting keys
- Station-to-Station protocol and Diffie-Hellman key exchange
- Each packet includes a *nonce* (unique identifier)
  - Prevents replay attacks
- *Message authentication code*
  - *Basically a verification hash*

Assigning File Keys

- What granularity should be used for keys?
  - File, directory, file system?
  - Small grain limits potential exposure
  - Large grain eases key management
- Key files assigned on per-directory basis
  - Physical data locations on disk favor this
  - Each directory has *keyfile*, contains K_u and K_g

Handling Keys Efficiently

- Key acquisition is a expensive
  - Cache decrypted keys
  - Overlap with disk I/O to mask seek time
  - Cannot overlap write/lookup, but not usually needed anyway
- Creating directories has no caching
  - Prefetch keys from token on binding
  - Store extra keys on token
  - Background daemon to obtain more keys

Departure and Return

- Two reasons for no token response
  - User is absent
  - Dropped packet
- Must recover from packet drop
  - Channel should be uncongested, so round-trip time is stable
    - No exponential backoff used
  - Retry if no response in twice expected time
  - Three maximum attempts

Departure and Return

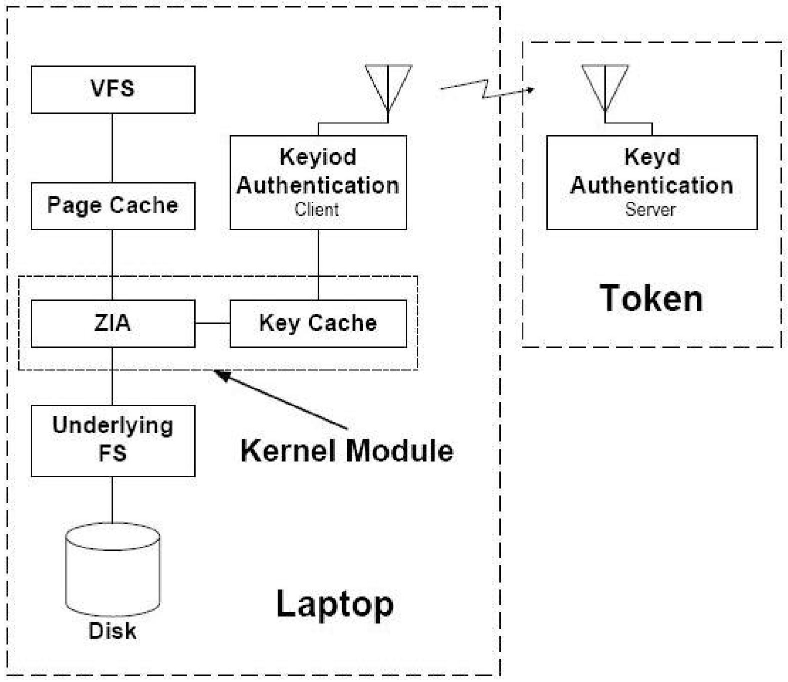
- After 3 failed attempts, user is declared absent
  - Remove all file name caches
  - Encrypt all page cache in-place
- On return, restore state quickly
  - Return caches, mark them valid
  - Must finish before the user is ready to resume
- Larges possible cache can be encrypted within 5 seconds
  - Compromising in this time would be impressive
- Disk operations will block while user is away
- Fixes beacons could be used in buildings

Overview

- Introduction
- Design of System
  - Encryption Model
  - Authentication Token
  - File Ownership
  - Departure and Return
- ➢ Implementation
- Evaluation

Implementation

- **In-kernel module**
  - **Provides cryptographic I/O**
  - **Manages keys**
  - **Poll for token presence**
- **User space authentication system**
  - **Client/server for token requests**
  - **Key generation**
- **Linux 2.4.10**
- **FiST for stackable file system**



This figure shows ZIA's design. The kernel module handles cryptographic file I/O. The authentication client and server manage key decryption and detect token proximity. A key cache is included to improve performance.

Figure 3: An overall view of ZIA

Kernel Module - VFS

- Virtual File System (VFS) abstracts file system code in kernel
- Stackable file systems separate upper and lower halves
  - Perform page transformations
  - CryptFS is part of 2.6 series, from FiST package

Kernel Module - Encryption

- Kernel module encrypts file pages and names
- Encrypted with Rijndael (AES)
  - Good performance and it's a standard
  - File size is preserved
  - Size of file names not preserved, Base-64 encoded so names are not rejected
- Module responsible for prefetching fresh keys for directory creation
- Module manages storage of encrypted keys, not visible in directory listing

Kernel Module – Token Polling

- Polling to make sure token still present
- Polling cannot be done by user process
- Polling period must be longer than small multiple of round-trip time
  - Period selected is 1 second
  - Poll message consists of a *nonce*
- When user is absent, laptop is secured
  - Cached data is encrypted, marked invalid
  - File keys are flushed (zeroed)
  - Added flag to page structure to mark encrypted pages
- On return, decrypted file keys refetched
  - Pages are decrypted, marked valid
  - Overlap key fetch and decryption to improve latency

Authentication System

- Authentication system implemented in user space
  - Communications encrypted by session key plus *nonce*
  - Communicate with UDP
- Declare user absent after three missed messages
  - Tunable option
- Session establishment is most taxing operation
  - Infrequent, so token can be on low-power hardware

Overview

- Introduction
- Design of System
  - Encryption Model
  - Authentication Token
  - File Ownership
  - Departure and Return
- Implementation
- ➢ Evaluation

Evaluation

- What is the cost of key acquisition?
- What overhead does ZIA impose?
  - What contributes to this overhead?
- Can ZIA secure a machine quickly enough to prevent attack?
- Can ZIA recover state before the user resumes work?

Test Machines

- IBM Thinkpad 570
  - 128MB RAM
  - PII 366MHz
  - 6.4GB IDE hard drive, 13ms seek time
- Compaq iPAQ 3650 as token
  - 32MB RAM
  - 802.11 wireless connection > 1Mbps
  - 128 bit key lengths

Key Acquisition

- Must compare key acquisition time to typical file access time
  - Measure elapse time between kernel request for key decryption and delivery of key
  - Average measured time 13.9ms
  - std. deviation 0.0015
- Empirical seek rate is comparable to seek rate of disk drive

ZIA Overhead

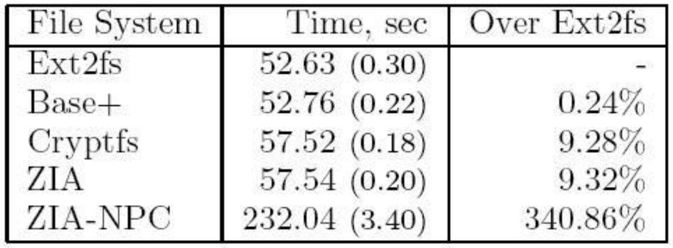
- Used benchmark scheme similar to Andrew Benchmark, which focuses on compiler performance
  - Copying kernel source tree
  - Traversal of source tree
  - Compiling of kernel source
- Perform 3 I/O-intensive tests to compensate for focus on compilation
  - Directory creation – cost of key creation
  - Directory Traversal – cost of acquisition
  - Tree copying – cost of data encryption

Andrew Benchmark Results

- Compare to ext2, Base+, ZIA, ZIA-NPC
  - Base+ is a null stacked file system
  - Cryptfs is a sample encrypted fs
    - For fair comparison, replace Blowfish with Rijndael
  - ZIA-NPC obtains keys on each disk access; no caching or prefetching
- 20 runs for each experiment
  - Compile source at different location

Andrew Results (cont'd)

- Base comparison results
  - ext2 registers base performance
  - Base+ shows stacked file system penalty
  - Cryptfs shows overhead for encryption
  - ZIA incurs both penalties
- Key caching is critical
- ZIA imposes penalty less than 10%



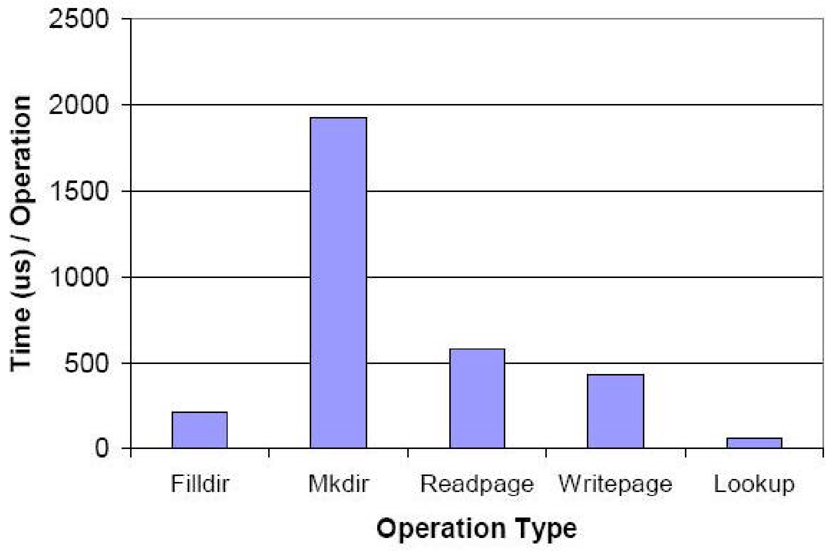
File System	Time, sec	Over Ext2fs
Ext2fs	52.63 (0.30)	-
Base+	52.76 (0.22)	0.24%
Cryptfs	57.52 (0.18)	9.28%
ZIA	57.54 (0.20)	9.32%
ZIA-NPC	232.04 (3.40)	340.86%

This shows the performance of Ext2fs against five stacked file systems using a Modified Andrew Benchmark. Standard deviations are shown in parentheses. ZIA has an overhead of less than 10% in comparison to an Ext2fs system and performs similarly to a simple single key encryption system, Cryptfs.

Figure 4: Modified Andrew Benchmark

Andrew Results (cont'd)

- Most of the 28 major fs operations are same time as Base+
- 5 operations are affected by ZIA
- All slowdowns are related to key management



This shows the per-operation overhead for ZIA compared to the Base+ file system. Writing and reading directory keys from disk is an expensive operation, as is encrypting and decrypting file pages.

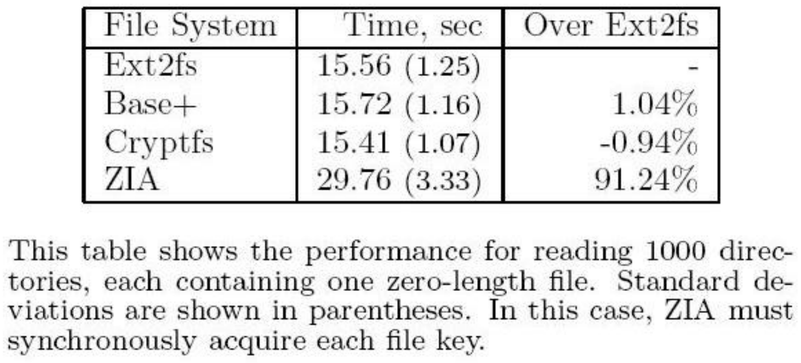
Figure 5: Per-Operation Overhead

I/O Benchmark Results

- Create 1000 directories, each with a zero-length file
  - Force creation of 1000 keys
  - Extra disk write for each file
  - Filename is encrypted too
- Reading from 1000 directories
- Copy Pine 4.21 source, 40.4MB, 47 directories

I/O Results (cont'd)

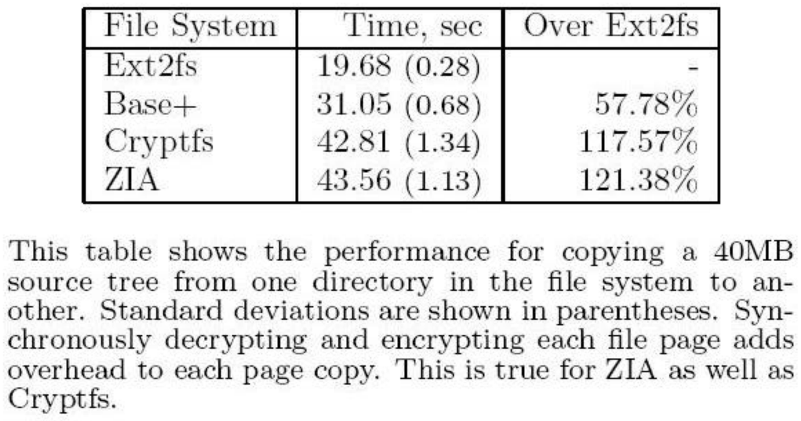
- ZIA has a large overhead
  - Each lookup is cache-cold
  - Key lookup time 14ms
- Rebooted between tests to empty cache
- Faster than expected
  - Aged file systems might not optimize file placement as well
  - Considering moving all keys into a specific location, read in batches
- Cryptfs and ZIA have to decrypt, copy, encrypt



File System	Time, sec	Over Ext2fs
Ext2fs	15.56 (1.25)	-
Base+	15.72 (1.16)	1.04%
Cryptfs	15.41 (1.07)	-0.94%
ZIA	29.76 (3.33)	91.24%

This table shows the performance for reading 1000 directories, each containing one zero-length file. Standard deviations are shown in parentheses. In this case, ZIA must synchronously acquire each file key.

Figure 7: Scanning Directories



File System	Time, sec	Over Ext2fs
Ext2fs	19.68 (0.28)	-
Base+	31.05 (0.68)	57.78%
Cryptfs	42.81 (1.34)	117.57%
ZIA	43.56 (1.13)	121.38%

This table shows the performance for copying a 40MB source tree from one directory in the file system to another. Standard deviations are shown in parentheses. Synchronously decrypting and encrypting each file page adds overhead to each page copy. This is true for ZIA as well as Cryptfs.

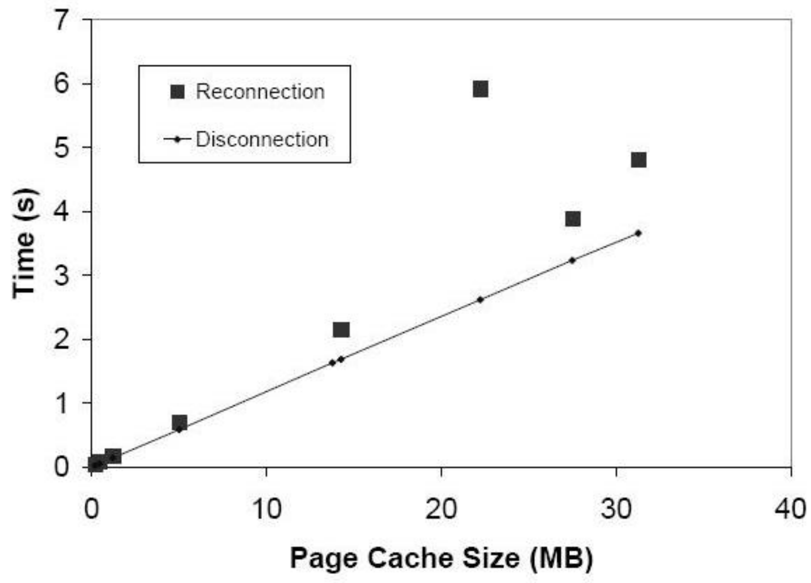
Figure 8: Copying Within the File System

Departure and Return

- For security, ZIA must encrypt all file page data soon after departure
- ZIA must restore machine state before user resumes work
- To measure dis/reconnect time
  - copy source directories, remove token, returned token

Departure and Return (cont'd)

- **Line shows time to secure laptop**
  - **Linear with amount of data**
- **Big blocks are time to restore state**
  - **Mostly linear with variations because of key fetching**
- **Window of 5 seconds is too short for an exploit to take place**
- **User walking back to his seat once in range should be > 6 seconds**



This plot shows the disconnection encryption time and reconnection decryption time. The line shows the time required to encrypt all the file pages when the token moves out of range. The blocks show the time required to refetch all the cached keys and decrypt the cached file pages.

Figure 9: Disconnection and Reconnection

Conclusion

- Laptops are still vulnerable to theft
- Cryptographic file systems do not offer sufficient protection
  - Long-term authority
  - Closing this vulnerability burdens user
- ZIA protects without undue strife or speed penalty