RANDY: Authentication and Encryption using One-Time Pad

Shoukat Ali, Ryan Henry, Joel Reardon, Rei Safavi-Naini

Department of Computer Science, University of Calgary

shoukat.ali@ucalgary.ca

Feb. 26, 2021

Outline

Background

- G. Vernam invented a cipher in 1917 for teletype communication
- G. Vernam and J. Mauborgne (U.S. Army Captain) developed one-time pad (OTP)
- The famous *hot line* between the White House and the Kremlin
- The pencil-and-paper versions used in diplomatic correspondence

Introduction

- Suppose $K, M,$ and C are the set of keys, messages, and ciphertexts, respectively
- In OTP cipher, encryption and decryption operations are performed as follows
  - In encryption, $M \oplus K = C$
  - In decryption, $C \oplus K = M$
  - Where $\oplus$ represents the *Exclusive OR* operation
- Encryption and decryption operations are very fast

Characteristic of OTP

- The OTP key should be truly random
- The OTP key should be at least of the same length as the message
- The OTP key should be used only once
- Only two copies of the OTP key should exist
- Both copies of the OTP are destroyed immediately after use

Example

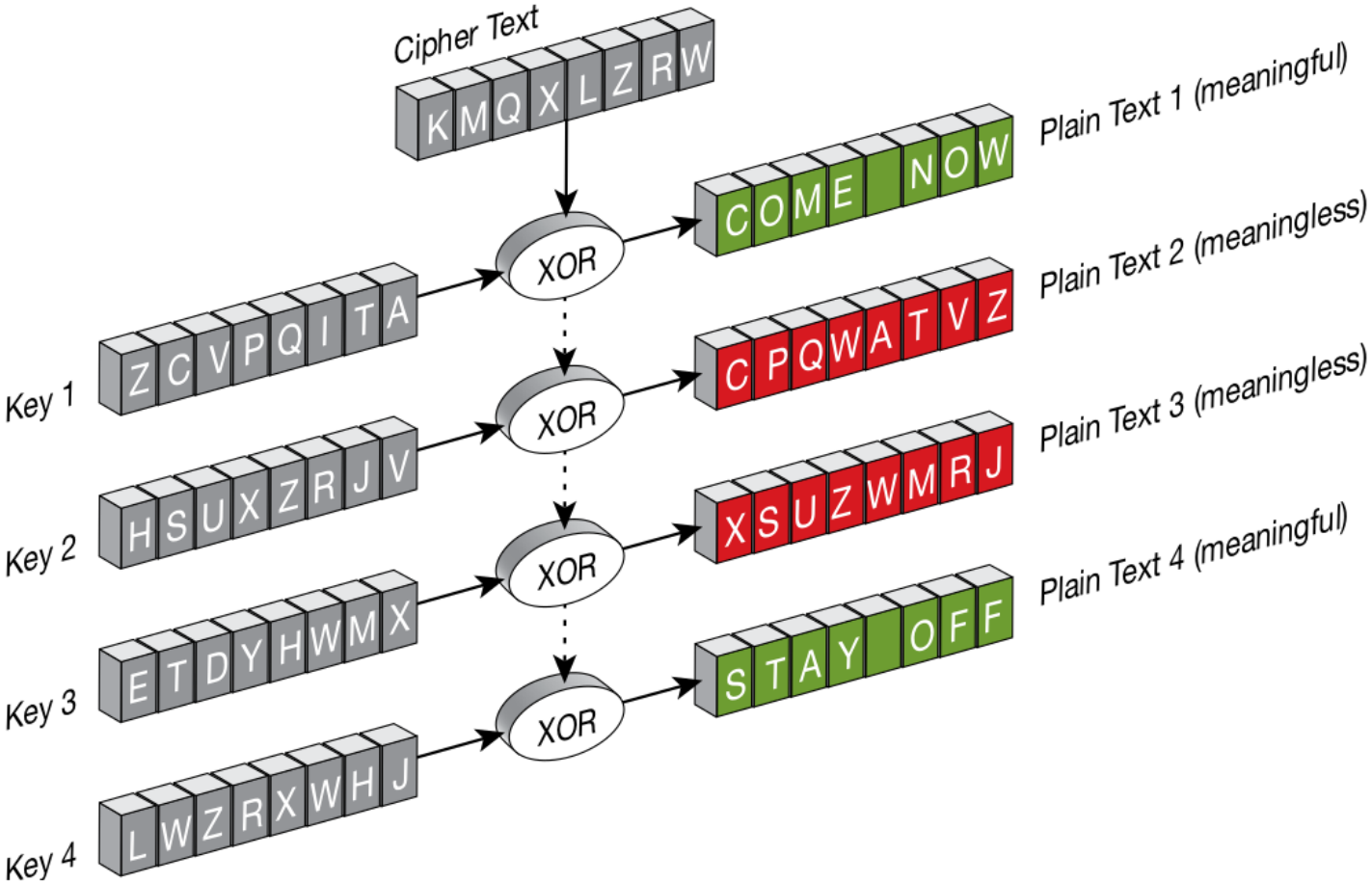

Figure: Human Language

OTP Security

- Is OTP secure?
- What is a secure cipher?
- Assume that the attacker is capable of seeing only the cipher-text

Shanon (Information-theoretic) Security

- Idea: Cipher-text should not reveal any information about the plain-text

Shanon (Information-theoretic) Security

- Idea: Cipher-text should not reveal any information about the plain-text

Definition-1

A cipher has perfect secrecy if $Pr[m|c] = Pr[m]$, for all $m \in M$ and $c \in C$, where M is the set of plain-text and C is the set of cipher-text.

Shanon (Information-theoretic) Security

- Idea: Cipher-text should not reveal any information about the plain-text

Definition-1

A cipher has perfect secrecy if $Pr[m|c] = Pr[m]$, for all $m \in M$ and $c \in C$, where M is the set of plain-text and C is the set of cipher-text.

Definition-2

A cipher has perfect secrecy if for all $m_0, m_1 \in M$ such that m_0 and m_1 are of same length and for all $c \in C$ we have

$$Pr\left[Enc(m_0, k) = c\right] = Pr\left[Enc(m_1, k) = c\right]$$

where $k \in K$ is chosen randomly.

Proof OTP

- Using Definition-2

Proof

$\forall m, c$

$$Pr\big[Enc(m,k) = c\big] = \frac{\text{No. of keys in } K \text{ such that } Enc(m,k) = c}{\text{Total no. of keys in } K}$$

we know that $k \oplus m = c \implies k = m \oplus c$

$$Pr\big[Enc(m,k) = c\big] = \frac{1}{\text{Total no. of keys in } K}$$

User-space and kernel-space

- User applications are executed in *user-space* and see a subset of machine's available resources
- An elevated system state where full access to all machine's resources is available is referred to as *kernel-space*

User-space and kernel-space

- User applications are executed in *user-space* and see a subset of machine's available resources
- An elevated system state where full access to all machine's resources is available is referred to as *kernel-space*

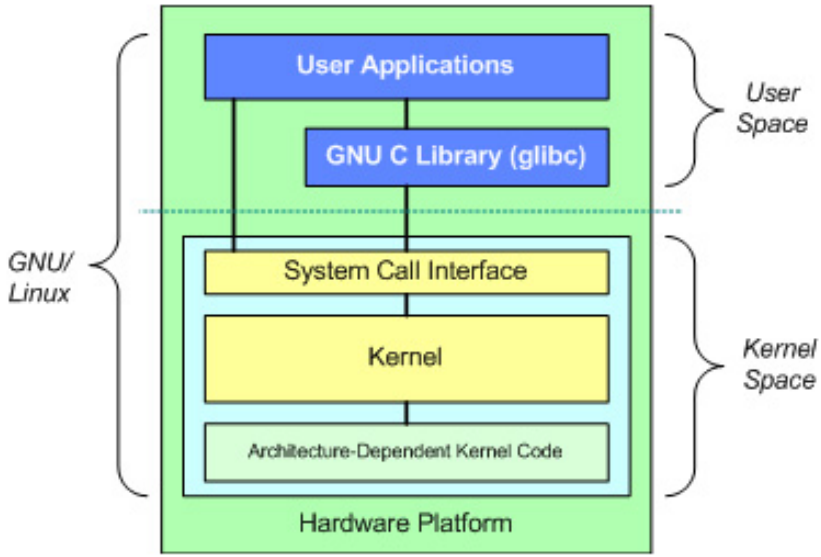

Figure: An overview of the user-space and kernel-space

User Applications

- Text editor, spreadsheet, word processing, audio and video players, web browser, etc. are some of the user applications
- Most of the applications that we use are executed in user-space

Why User-space?

- User-space programs run in protected memory

Why User-space?

- User-space programs run in protected memory
- User-space processes are not allowed to interfere with kernel memory or other user process memory

Why User-space?

- User-space programs run in protected memory
- User-space processes are not allowed to interfere with kernel memory or other user process memory
- User-space processes do not bring down the entire operating system if they crash

Why User-space?

- User-space programs run in protected memory
- User-space processes are not allowed to interfere with kernel memory or other user process memory
- User-space processes do not bring down the entire operating system if they crash
- User-space programs can easily be debugged

Why User-space?

- User-space programs run in protected memory
- User-space processes are not allowed to interfere with kernel memory or other user process memory
- User-space processes do not bring down the entire operating system if they crash
- User-space programs can easily be debugged
- But user-space processes have significant overhead when making system calls

Why Kernel-space?

- Kernel-space programs can handle interrupts

Why Kernel-space?

- Kernel-space programs can handle interrupts
- Kernel-space programs require less context switching

Why Kernel-space?

- Kernel-space programs can handle interrupts
- Kernel-space programs require less context switching
- Kernel-space programs have lower-level access to system resources

Kernel-space: Buts

- No GNU C library (glibc)
- Kernel-space programs can access the whole physical memory which implies no memory protection
- Kernel-space programs can crash the whole system
- Kernel-space programs debugging is not as easy as user-space application
- Kernel-space programs have no automatic clean-up

RANDY

Using the OTP, the RANDY performs the following operations in kernel-space

- Data integrity
- Authentication
- Encryption/Decryption
- Authenticated Acknowledgment

Data Integrity: Wegman-Carter MAC

Let (S, V) be a secure one-time MAC over $(K, M, \{0,1\}^n)$ and $F : K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ be a secure PRF.

Definition: Wegman-Carter MAC

$$WC\big((k_1, k_2), m\big) = \big(r, F(k_1, r) \oplus S(k_2, m)\big)$$

where $(k_1, k_2) \in K, m \in M$ and for random $r \in \{0,1\}^n$

Authentication Protocol: Goals

- To perform mutual authentication assuming that the server is trusted and secure
- To establish (shared) session_longterm_key that will serve as the long-term part for the Wegman-Carter (WC) key in a session
- To decide the pad/key for encryption/decryption and the short-term part for the WC key in a session
- To protect against an active network attacker
- To protect the server against the random exhaustion attack

Authentication Protocol: Assumptions

- The client is in hostile environment
- The client always initiates the connection
- Both the client and server always use fresh pad/key from the shared pool of randomness
- If authentication fails, then the server can return the pads/keys used during authentication to the shared pool of randomness

Authentication Protocol: In Picture

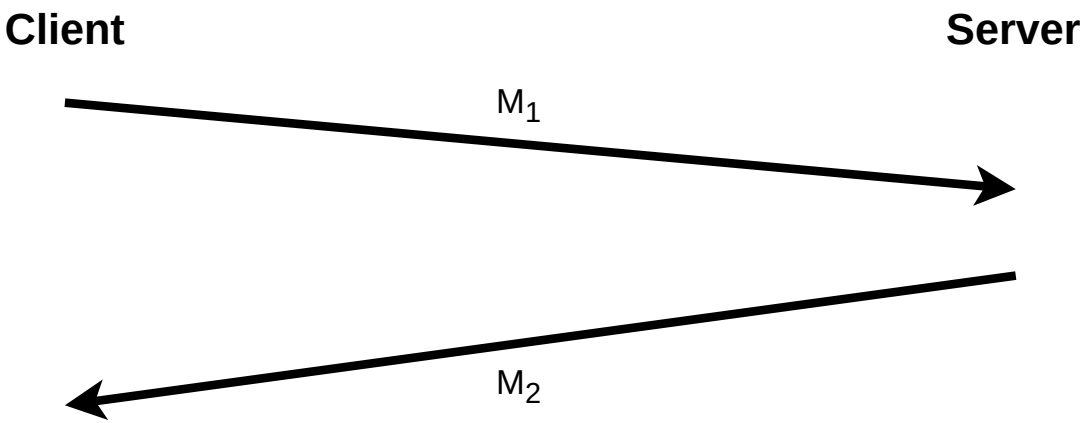

Figure: Mutual Authentication Protocol

Authentication Protocol: Some details

- Authentication is based on shared password and randomness pool
- Password is never transmitted on wire
- The Password-Based Key Derivation Function (PBKDF2) is used to derive a key from the password
- Wegman-Carter style of Message Authentication Code (MAC) is used of data integrity
- From the pool of randomness, both the communicating parties use one-time pads not only from its parity but also the other party.

Encryption/Decryption

The encryption and data integrity operations are perfromed as follows

- cipher-text $=$ plain-text $\oplus$ Pad
- $WC\big((k_1, k_2), m\big)$ where k_2 is the shared session_longterm_key, k_1 is short-term key, and m is the cipher-text in Wegman-Carter MAC

The data authenticity and decryption are performed accordingly

Authenticated Acknowledgment

- To restrict the (active) network attacker's capability by allowing at most some fixed amount of unacknowledged data in-flight per socket
- The sender transmits next data only after the reception and verification of authenticated acknowledgment (auth_ack) for in-flight data on that socket
- The receiver knows when to transmit auth_ack

Authenticated Acknowledgment: In Picture

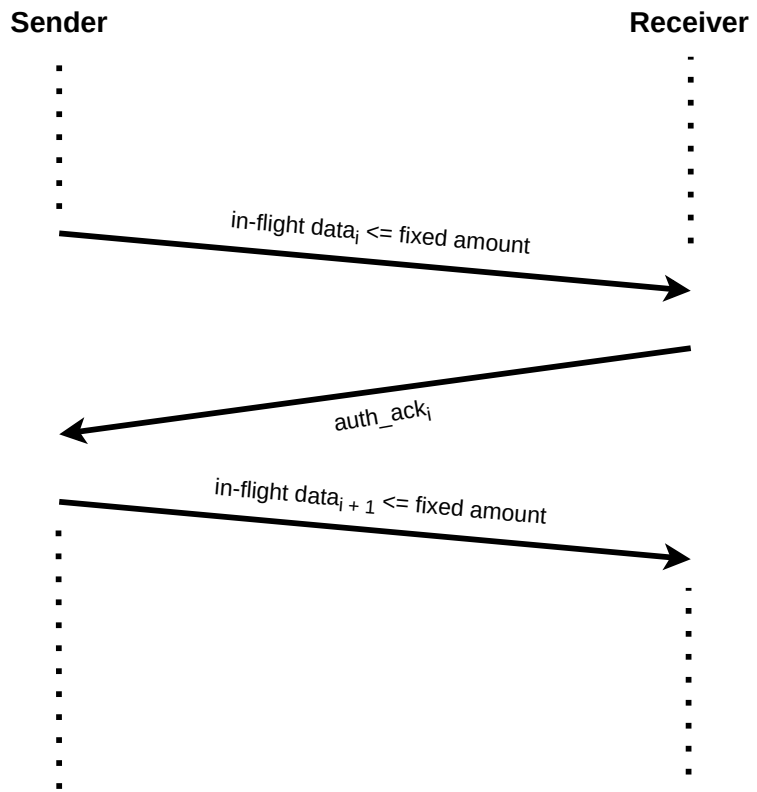

Figure: Overview of Data transmission and authenticated acknowledgment

Our Implementation: Overview

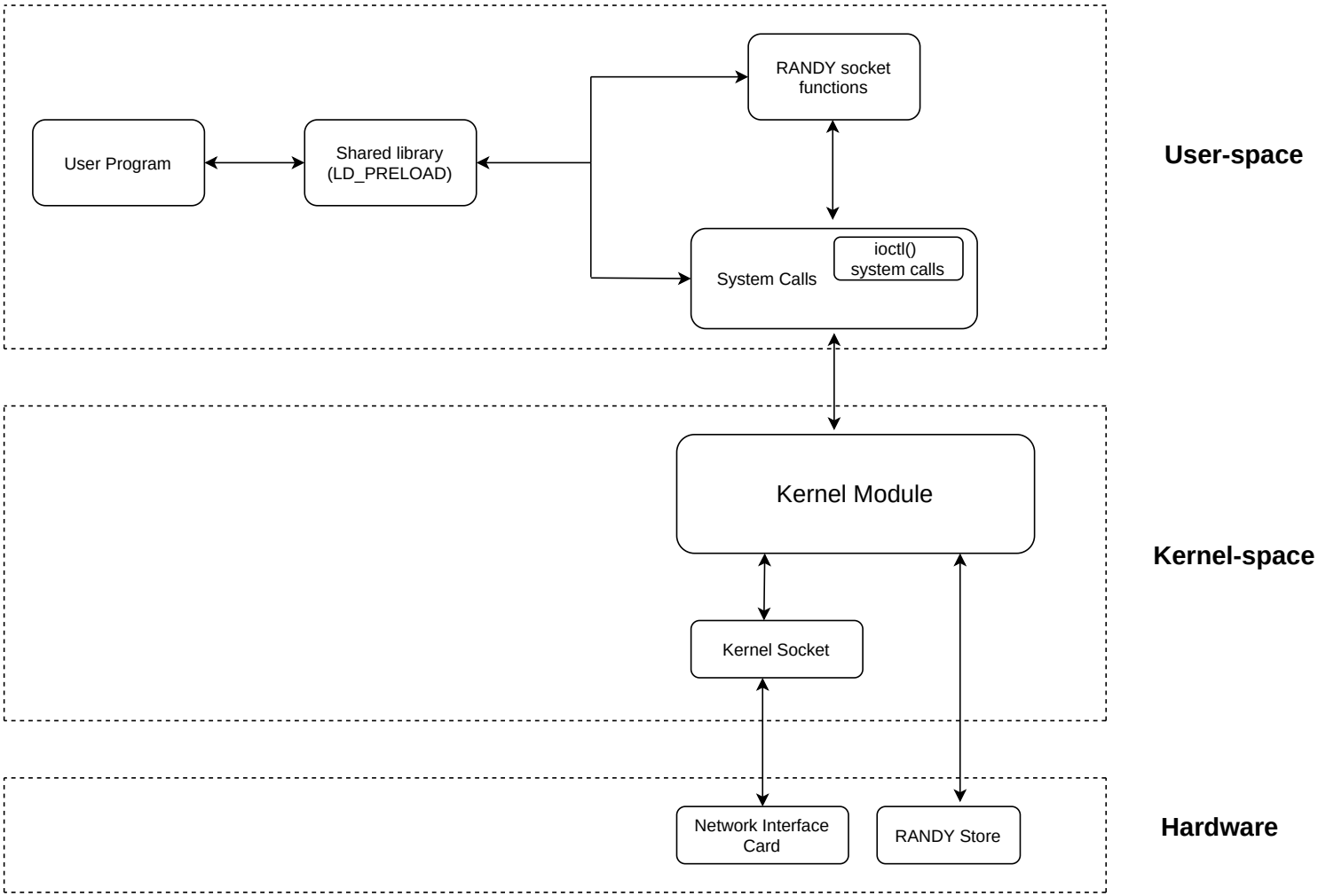

Figure: Overview of RANDY

Our Implementation: RANDY store

- A storage device that is filled with randomness
- The client and server use different parity
- For encryption, a communicating party (client or server) uses one-time pads from its parity
- for decryption, a communicating party (client or server) uses one-time pads from the parity of other party

Our Implementation: User-space

Some of the operations performed in user-space are as follows

- Loading the kernel module
- Taking user password and writing it in kernel-space
- Executing user application
- System calls to RANDY socket

What is LKM?

- The Linux kernel supports Loadable Kernel Module (LKM) mechanism
- The LKM provides the following advantages
  - Loading module into base kernel at run-time
  - Save kernel memory by loading module when needed and unloading when not needed
- Hence, new features/code can be added while the operating system is running

Our Implementation: Kernel-space

Some of the operations – other than data integrity, authentication, encryption/decryption, and auth_ack – performed by our kernel module are as follows

- Copying user message into kernel-space
- Fetching and managing the One-Time pad
- Managing (RANDY) socket
- Purging sensitive information

Thanks for your attention!
Questions?