

# Устная часть доклада

## Slide 1: Часть 1 - Введение

Всем привет! Меня зовут Фам Нгия.

Сегодня я расскажу об очень важной теме в области компьютерных наук - Шифрование данных в компьютерах: от ASCII до Unicode и ошибки кодировки.

## Slide 2: План доклада

У нас вообще 7 этапов:

1. Введение
2. Принцип работы систем кодирования символов
3. Популярные ошибки кодирования и их последствия
4. Проблемы безопасности при шифровании данных
5. Практические примеры проблем шифрования
6. Состояние и будущее шифрования данных
7. Заключение

## Slide 4 - Почему шифрование так важно?

В жизни Иногда мы встречаемся с некорректным отображением текста, например, вот такие.

Это следствие неправильного или несовместимого шифрования между различными системами.

Шифрование данных не только помогает компьютерам понимать текст, но и обеспечивает возможность обмена данными между различными системами.

Без стандартной системы шифрования у нас было бы множество проблем с отображением символов, потерей данных или даже рисками безопасности.

## Slide 5 - История развития шифрования данных: от ASCII до Unicode

кодировщик ASCII был впервые представлен в 1963 году ASA -

для представления текста в виде двоичного числа, -

но он поддерживает только 128 символов, в основном для английского языка.

он не мог в полной мере поддерживать другие языки, такие как русский, вьетнамский, японский...

Чтобы преодолеть это, в 1991 году Unicode был разработан как общий кодировщик -

неторговой организацией Unicode Consortium с участием Apple, Microsoft, IBM, Google, Adobe, Oracle и многих других.

Он может хранить сотни тысяч символов со всех языков мира.

теперь Unicode - особенно UTF-8 - стал самым популярным стандартом в компьютерных системах

## Slide 6: Phần 2: Принцип работы систем кодирования символов

### Slide 7 - ASCII – Первый кодировщик символов

💡 Одной из первых и наиболее популярных систем кодирования была ASCII (American Standard Code for Information Interchange).


#### 📌 Cách hoạt động của ASCII

- ASCII - это 7-битная кодировка, которая может содержать 128 символов, включая:
  - Английские буквы (A-Z, a-z).
  - цифры (0-9).
  - Специальные символы, такие как @, #, \$, %, &, .
  - Управляющие символы, такие как Enter, Tab, Backspace.
- Потом ASCII расширилась до 8-битного (Extended ASCII), чтобы она содержала 256 символов и добавил некоторые специальные символы.

## Giới hạn của ASCII

- ASCII поддерживает только английский язык, невозможно представить специальные символы других языков.

## Slide 8 - Unicode – Глобальные решения

 Поскольку ASCII не удовлетворила потребность во всех языках, вместо него в 1991 году появился Unicode.

## Cấu trúc của Unicode


Unicode использует набор плоскостей (planes) для хранения символов:

- **BMP (Basic Multilingual Plane)** – Основная плоскость (от U+0000 до U+FFFF (т.е. от 0 до 65 535 в десятичной системе счисления)) содержит наиболее популярные символы. Группы символов классифицируются в соответствии с различными цветами на изображении: ...
- **Supplementary Planes** – В Unicode есть не только BMP, но и дополнительные плоскости для добавления специальных символов, таких как эмодзи, древние символы и редкие языки. На картинке выше показан первый дополнительный план

## Slide 9 - Варианты Unicode


В Unicode есть различные варианты кодирования для оптимизации хранилища:

### 1 UTF-8 (8-bit Variable-Length Encoding) - 4 bytes


 Наиболее часто используется в веб-системах и программном обеспечении.

 Совместит с ASCII

 Сохранит объем при хранении текста на английском языке.

 Не может представить символы, отличные от ASCII (из-за того, что занимают более 4 байт).

### 2 UTF-16 (16-bit Encoding)

 Поддерживает весь Unicode , используя минимально 2 байта.

✓ Подходит для систем, которым необходимо обрабатывать символы отличные от ASCII

✗ Не обратно совместит с ASCII. (Если система поддерживает только ASCII, она не сможет точно считывать данные в формате UTF-16)

### 3 UTF-32 (32-bit Encoding)

✓ Каждый символ имеет фиксированную длину в 8 байт, что ускоряет поиск.

✗ Тратит много памяти, меньше используется на практике.

## 📌 Slide 10 - Tương tác với lớp học Обсуждение

💬 Hỏi cả lớp: Как вы думаете?

- "Может ли Unicode полностью заменить ASCII? Почему некоторые системы все еще используют ASCII?"
- "Как вы думаете, почему UTF-8 используется чаще, чем UTF-16 и UTF-32?"

### 🗨 Thảo luận:

- ASCII еще используется во многих старых системах, поскольку он легкий и простой.
- UTF-8 более популярен из-за совместимости с ASCII и экономии памяти для английского текста.

## Slide 11 - Ошибки кодирования и их последствия

### Slide 12- Потеря данных при переводе между системами шифрования

◆ Почему так?

Эта ошибка возникает при переводе между двумя несовместимыми системами кодирования, например Unicode на ASCII, поскольку ASCII не полностью поддерживает символы, которые есть в Unicode.

📌 Пример:

Допустим, вы сохраняете текстовый файл, содержащий русские символы, но система поддерживает только 7-разрядный формат ASCII.

- При сохранении файла эти символы отсутствовали в таблице ASCII.
- Система может заменить их другими символами или даже проигнорировать.
- Когда вы снова откроете файл, исходный символ будет потерян навсегда.

## Slide 13 - Phân tích lỗi và đưa ra giải pháp

 Некоторые причины потери данных при переводе шифрования:

**1** Использовать кодировки, которые не полностью поддерживают символы (например, от Unicode до ASCII).

**3** Использовать программное обеспечение, которое не поддерживает Unicode полностью.

 Решение против этой ошибки:

✓ Всегда используйте UTF-8 для хранения текста вместо ASCII.

✓ Проверьте совместимость системы перед импортом /экспортом данных.

## Slide 14 - Отображение неправильных символов (Mojibake)

 Что такое **Mojibake** ?

Mojibake (文字化け) - японский термин, означающий "искаженный символ".

Эта ошибка возникает, когда текст зашифрован с помощью кодировщика, не расшифрован с помощью другого несовместимого кодировщика.

 Пример:

Допустим, вы сохраняете файл с шифрованием UTF-8, но когда вы открываете файл в ПО, которое поддерживает только Windows-1252,

то система может не понимать символы Unicode и отображать их неправильно, например:

## Slide 15 - nguyên nhân và giải pháp

◆ Причины ошибки **Mojibake**:

1 Текстовый файл закодирован одним кодировщиком, но считается другим кодировщиком.

3 Перевод между несовместимыми кодировщиками без надлежащих правил.

💡 Решения против ошибки **Mojibake**:

✓ Используйте UTF-8 в качестве стандарта по умолчанию, особенно в Интернете и системах хранения текста.

✓ Проверяйте и устраняйте ранние ошибки отображения с помощью инструментов тестирования шифрования, таких как `iconv` или `chardet`.

## Slide 16 - Сравнение ошибок Mojibake и потери данных

Вообще мы поняли разницы между ними. **Mojibake** просто неправильно причитает текст, не потеря исходные данные. А ошибки потеря данных полностью потеряли данных и могут заменить их странным символам.

Критерий	Ошибки Mojibake	Ошибки потери данных
 Причина	Чтение данных с помощью кодировщика отличается от кодировщика для сохранения	Сохранение данных в кодировках, которые не полностью поддерживают символы
 Упругость	Может быть исправлено, если знаете исходный кодировщик	Невозможно восстановить, если данные были потеряны
 Исходные данные	Еще есть, но неправильно показывается	Часть данных полностью утеряна
 Оознавательные знаки	Появляются странные персонажи	Пропущенные символы заменены на? или □
 Как исправить	Определите правильный кодировщик и перечитайте файл еще раз	Используйте UTF-8 или Unicode для правильного хранения данных

## Slide 17. Проблемы, связанные с безопасностью при шифровании данных

## Slide 18. Атака, основанная на неправильном шифровании

### Lỗi gì xảy ra?


- Когда разные части системы не совместимы о том, как обрабатывать шифрование данных, -

хакеры могут воспользоваться этой разницей, -

чтобы обойти проверки безопасности, -





внедрить вредоносное ПО или нарушить логику работы программы.


### Примеры:

 Некоторые символы Unicode имеют разное представление, но отображаются одинаково.

Это может быть использовано хакерами для обхода проверок безопасности.

### Пример 1:

- Символ  в Unicode это может быть представлено двумя способами:
  - **U+00E9** →  (один обычный символ)
  - **U+0065 + U+0301** →  (символ  + знак)

 В этом примере хакеры могут внедрять вредоносные программы, оставаясь незамеченными.

### Пример 2:

**Предположим, что в системе есть следующий этап проверки входа в систему :**

```
if username == "admin":  
    print("Нельзя проходить!!!")
```

◆ Эта система запрещает кому-либо вводить "admin" в поле имени пользователя.

## Hacker có thể nhập gì để bypass?

Вместо ввода "  " хакер вводит:

admin

В этом числе:

- Символ `n` (U+1D5FB) это версия буквы `n` из математического кодировщика **Mathematical Sans-Serif**).

👉 Система проверяет условие `username == "admin"` и не распознает "`admin`" как "`admin`", потому что `n ≠ n`.

- Мы смотрим глазом, что "`admin`" и "`admin`" выглядят совершенно одинаково и разрешает отправителю работать

✅ **Kết quả:** хакеры проходят проверку на безопасность.

📌 Slide 19 - Способы защиты :

✓ Стандартизация данных в Unicode перед обработкой (**Unicode Normalization**).

✓ Используйте библиотеку, которая поддерживает безопасную обработку в формате Unicode, например `unicodedata.normalize ()` в Python.

✓ Проверьте и удалите ненужные символы перед сохранением или обработкой.

## Slide 20 - SQL Injection приводит к ошибкам шифрования

📌 Lỗi gì xảy ra?

- **SQL-Injection** - это одна популярная форма атаки в Интернете, - при которой хакеры вставляют вредоносные инструкции SQL во входные данные для атаки базы данных.

- Если система неправильно обрабатывает шифрование, - хакеры могут использовать специальные символы в **Unicode**, чтобы обойти систему проверки входных данных и - выполнить вредоносный SQL-код.

📌 Пример: Мы уже узнали о SQL Injection



//Пусть в системе есть SQL-запрос, используемый для проверки данных логина:

```
SELECT * FROM users WHERE username = 'admin' AND password = 'password123';
```

//хакер введет в поле username следующим образом:

```
admin' OR '1'='1
```

//то SQL-запрос будет:

```
SELECT * FROM users WHERE username = 'admin' OR '1'='1'; //TRUE
```

//SQL вернет все строки в таблице users, потому что это условие выполняется.

🔥 **Vấn đề:** Пусть в этом случае у нас система против SQL-Injection, - которая ищет и запрещает символы, такие как "OR".-

Но хакеры могут использовать тот же символ Unicode, - чтобы обойти процесс проверки.

💡 Они могут так делать:

- Символ "O R" символ " O R " - это не обычный символ ASCII, а вариант в Unicode ( `FULLWIDTH OR` – `U+FF4F U+FF52` ).
- Система проверки не узнает этот вариант Unicode и запрещает его, хакеры могут пройти проверку безопасности!

## 📌 Slide 21 - Способы защиты:

✓ Всегда используйте **Prepared Statements** вместо того, чтобы связывать строки напрямую в SQL.

✓ Блокируйте все нежелательные символы Unicode во входных данных

✓ Используйте безопасный механизм защиты вместо фильтрации символов по черному списку.

## 📌 Ví dụ mã an toàn sử dụng Prepared Statements trong Java:

```
String query = "SELECT * FROM users WHERE username = ? AND password = ?";
```

```
PreparedStatement stmt = connection.prepareStatement(query);
```

```
stmt.setString(1, username);  
stmt.setString(2, password);  
ResultSet rs = stmt.executeQuery();
```

## 5. Slide 22 - Проблемы из-за ошибок шифрования

🔍 Ошибки в процесс шифрования данных может иметь серьезные последствия на практике

Сегодня мы разберем два типичных случая:

### 1 Slide 23 - Ошибка в избирательной системе США (Florida, 2000)

#### 📌 Tóm tắt sự cố

- в 2000 году в США, Florida стала штатом, который решил исход выборов между **George W. Bush - Губернатор Texas** и **Al Gore - Вице-президент**.
- Во это время возникли проблемы с электронной системой голосования, - в результате чего было удалено более 18 000 голосов выбора
- Одна причина считает ошибкой в кодировке символов, -

из-за которой система неправильно считывает информацию из голосов выбора.

#### 📌 Nguyên nhân kỹ thuật

- Система регистрирует голоса, используя кодировщик, - который не полностью поддерживает Unicode
- поэтому При переводе между различными кодировщиками - некоторые символы кодируются неправильно, - что приводит к ошибке при анализе результатов голосования.

#### 📌 Hậu quả

- Из-за ошибки при подсчете голосов - результаты двух кандидатов отличали друг от друга только 537 голосов, что привело к недельным спорам.

- Если система шифрования правильно работает, -

то количество голосов, возможно, было подсчитано более точно.

## 2 Slide 24 - Ошибки отображения эмодзи на Facebook / Twitter

### 📌 Tóm tắt sự cố

- В 2014, **Facebook** и **Twitter** встречали с проблемами отображения эмодзи, когда некоторые значки стали в "?" или "□".
- Пользователи сообщают, что когда они вводят эмодзи на iOS или Android, -

некоторые другие платформы отображаются неправильно.

### 📌 Nguyên nhân kỹ thuật

- UTF-8 поддерживает эмодзи только начиная с Unicode 6.0
- Старые устройства или приложения не распознают новые эмодзи, поскольку они не были обновлены до более новой версии Unicode.
- Unicode 6.0 (2010) : 😂 (Face with Tears of Joy), теперь видны большинство устройств.
- Unicode 15.0 (2022): 🤔 (Melting Face), если вы отправите его на старое устройство (например, поддерживающее только Unicode 12.0), -

оно может отображать □,? или вообще ничего не отображать.

### 📌 Hậu quả

- Пользователи смущены, потому что эмодзи не работают на разных платформах
- Некоторые приложения для чата сталкиваются с ошибками при обработке эмодзи в сообщениях.
- Технологические компании должны были обновлять систему для поддержки нового Unicode .

## Slide 25 - Состояние и будущее шифрования данных

🔍 Шифрование данных прошло несколько этапов развития, от ASCII до Unicode, -

и сейчас Unicode – особенно UTF-8 – становится самым популярным стандартом во всем мире.

## Slide 26- Популярные в настоящее время стандарты шифрования

### ◆ UTF-8 – Стандарты кодирования в Интернете

📌 Теперь UTF-8 приходится более 97% веб-сайтов по всему миру.

- **мног Преимуществ:**

- ✓ Обратная совместимость с ASCII

- (nếu chỉ dùng ký tự ASCII thì chỉ cần 1 byte).

- ✓ Большая экономия памяти по сравнению с UTF- 16 и UTF-32..

- ✓ Хорошая поддержка на всех платформах, операционных системах и браузерах.

- ✓ Защита системы от популярных ошибок шифрования, таких как Mojibake.

### ◆ UTF-16 và UTF-32

- UTF-16 все еще используется в некоторых приложениях, особенно в Windows, Java и XML, но постепенно заменяется UTF-8.

- UTF-32 используется реже -

из-за большего потребления памяти-

без действительно больших преимуществ.

◆ Старые кодировщики, такие как ASCII, ISO-8859-1, Windows-1252, были почти устранены.

## Slide 27 - Направление будущего развития

📌 Существует ли система, которая заменяет Unicode?

- На данный момент у Unicode нет реальных конкурентов.

### Ứng dụng mã hóa trong hệ thống đa ngôn ngữ

- Растет спрос на кодировку символов многоязычную-

Unicode продолжит расширяться, чтобы поддерживать -  
редкие языки, математические символы, эмодзи и специальные символы.

- Если в будущем квантовые компьютеры или технологии искусственного интеллекта будут процветать, -

нам возможно понадобится новая система шифрования, -

более оптимизированная для высокоскоростной обработки.

## Slide 29 - Вывод

### 📌 Tại sao cần hiểu về mã hóa dữ liệu?

- Шифрование данных - основание любой компьютерной системы, помогая нам правильно хранить информацию и обмениваться ею.
- При отсутствии подходящего шифрования -

запись может отображаться неправильно, -

данные могут быть потеряны и, еще хуже, -

хакеры могут воспользоваться этой ошибкой для атаки на систему .

🚀 Таким образом, понимание принципов шифрования не только -

помогает обеспечить правильную обработку данных, -

но и помогает нам избежать ошибок и -

обеспечить безопасность наших систем.

## Slide 30 - Вопрос для экзамена (Trang cuối của bài thuyết trình)

Почему Unicode заменяет ASCII в большинстве современных компьютерных систем?

### 📌 Gợi ý trả lời:

- Unicode поддерживает все языки мира, в то время как ASCII поддерживает только английский.
- UTF-8 занимает больше места по сравнению со старыми системами шифрования, но при этом совместим с ASCII.
- Unicode становится общим стандартом, -

помогает минимизировать ошибки шифрования, -  
обеспечивая глобальную унификацию работы системы.

🎯 Спасибо вам за то, что выслушали!