



DevCon School

Технологии будущего

Практическое использование средств криптографии в .NET, Java и PHP

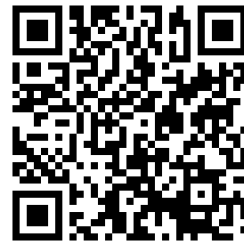
Владимир Кочетков
Positive Technologies

Positive Development User Group: разрабатываем безопасно вместе



POSITIVE DEVELOPMENT USER GROUP

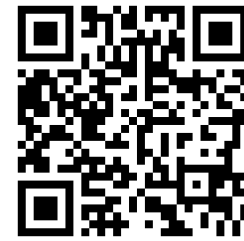
Открытое сообщество для разработчиков, которые хотят писать безопасный код.




 [Facebook](#)



 [YouTube](#)



 [SlideShare](#)

- Офлайн-встречи: митапы, семинары, воркшопы
- Вебинары
- Развитие бесплатной утилиты [Approof](#) 
- Связаться с нами: pdug@ptsecurity.com

Что?

Высокоуровневая
криптография

Случайные числа

Хэширование

Симметричное шифрование

Как?

Case-study: правильный
выбор средств
криптографии и их
эффективное использование

Почему?

Примеры реальных
уязвимостей

Когда и какую криптографию использовать?



RSnake
@RSnake

Maybe Alice and Bob should just kill Eve.
Sounds easier than all this math stuff.



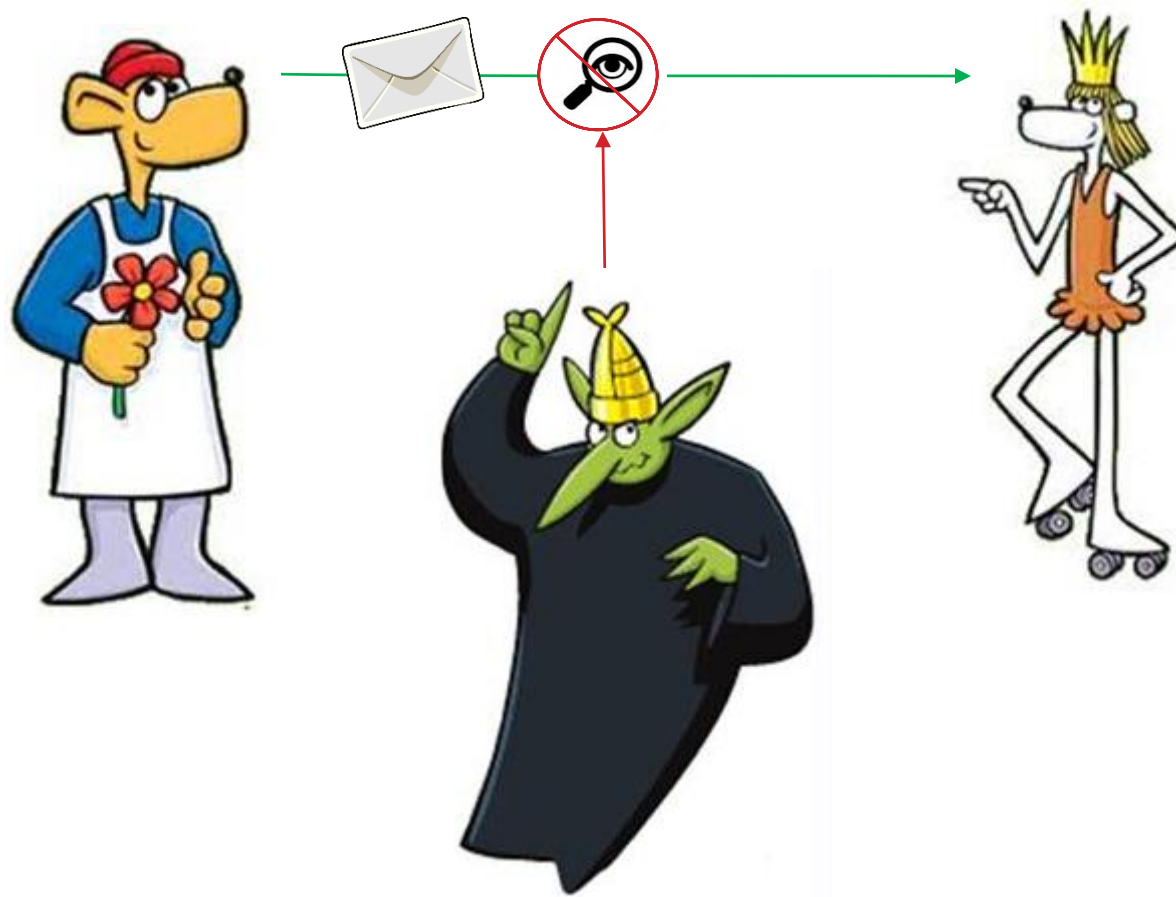
Применимость криптографии

Криптография обеспечивает
конфиденциальность, целостность и
аутентичность передаваемой или хранимой
информации и может также использоваться
для генерации случайных чисел

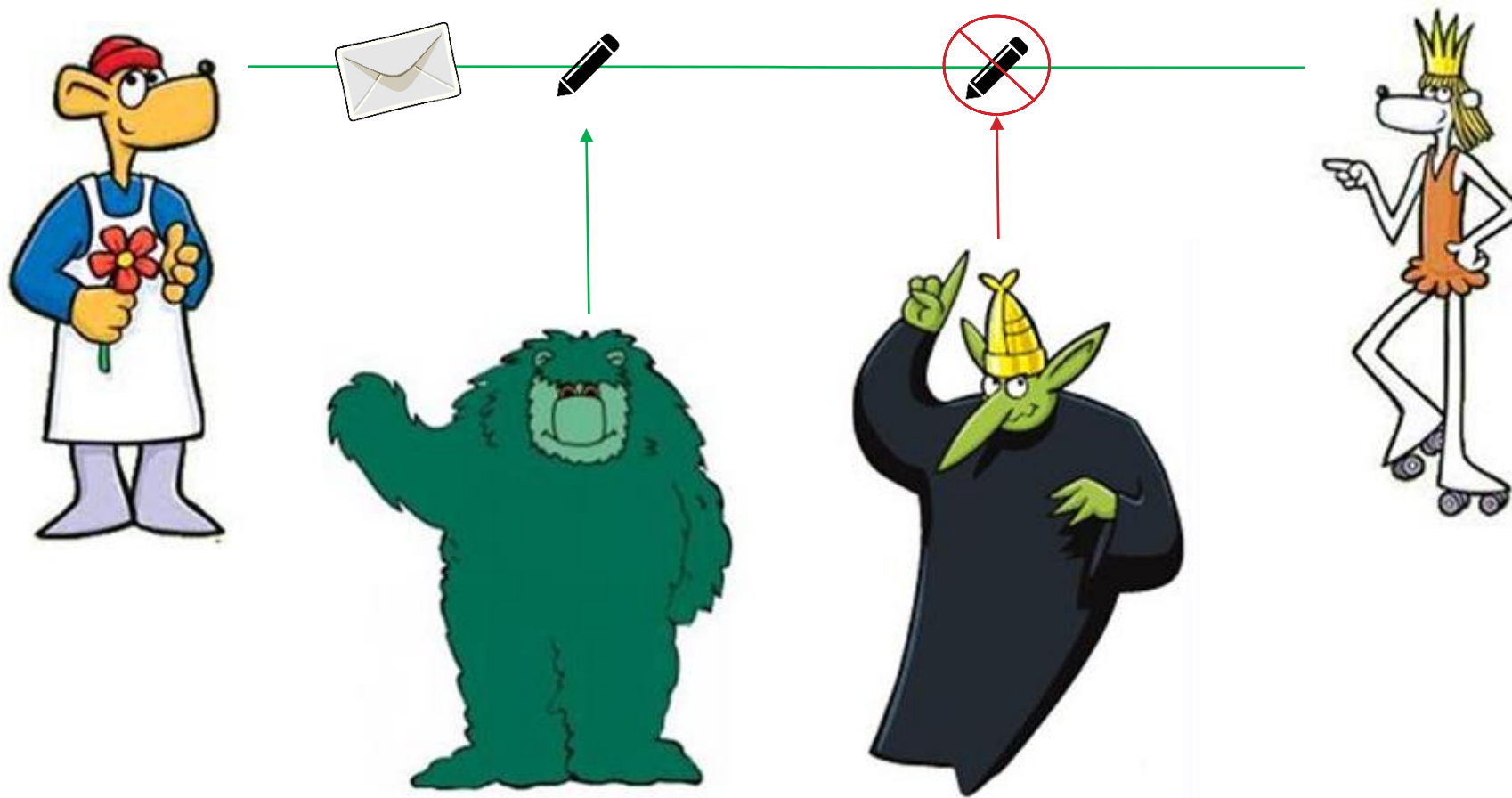
Muzzy in Cryptoland



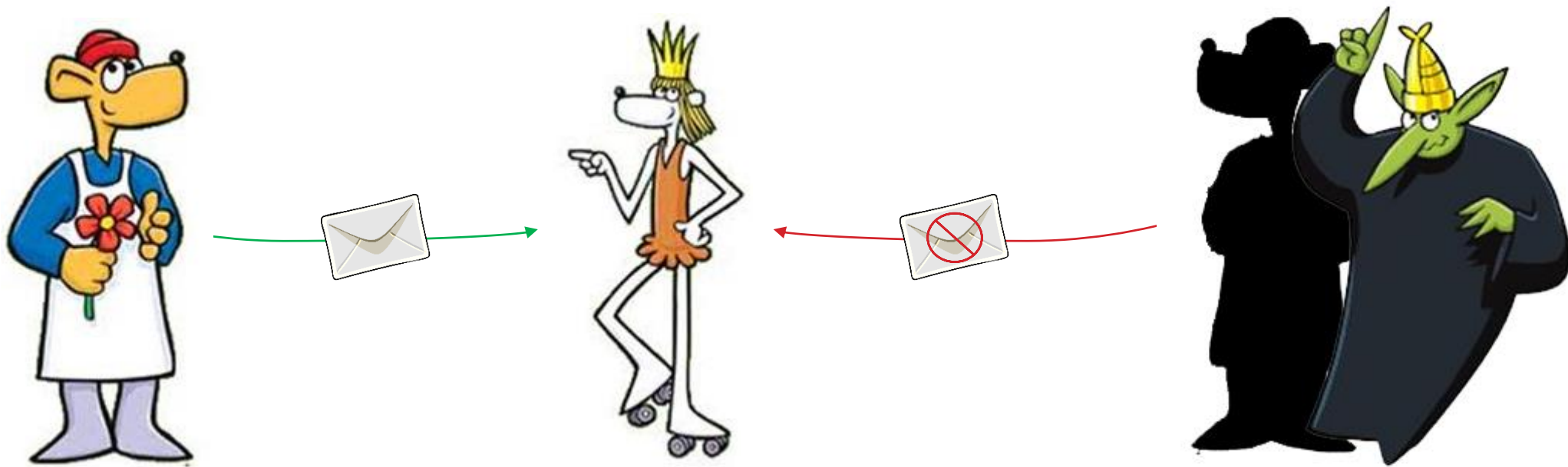
Конфиденциальность



Целостность



Аутентичность



Генерация случайных чисел



Законы об изобретении криптографии

Наблюдение Бэббиджа: каждый человек, познакомившийся с криптографией, стремится создать свой собственный устойчивый шифр.

Законы об изобретении криптографии

Уточнение Шнайера: создать свой собственный устойчивый шифр несложно. Сложно обеспечить его устойчивость перед всесторонним анализом независимыми экспертами.

Законы об изобретении криптографии

Критерий Тьюринга: устойчивым можно считать только шифр, подвергавшийся всестороннему анализу независимыми экспертами в течении нескольких лет.

Законы об изобретении криптографии

I-ый закон криптографии: не изобретайте собственные алгоритмы криптографии и не реализуйте сторонние!

Законы об изобретении криптографии

II-ой закон криптографии (только для экспертов): всё равно не изобретайте и не реализуйте!

Законы об изобретении криптографии

Дополнение Диффи-Хеллмана: ля-ля-ля, мы
тебя не слышим!!



Задача

Сравнить два байтовых массива

#msdevcon

Очевидный подход

```
1. public static bool Equals(byte[] a1, byte[] a2)
2. {
3.     if (a1.Length != a2.Length)
4.     {
5.         return false;
6.     }
7.
8.     for (var i=0; i < a1.Length; i++)
9.     {
10.        if (a1[i] != a2[i])
11.        {
12.            return false;
13.        }
14.    }
15.    return true;
16.}
```



Очевидный подход

```
1. public static bool Equals(byte[] a1, byte[] a2)
2. {
3.     if (a1.Length != a2.Length)
4.     {
5.         return false;
6.     }
7.
8.     for (var i=0; i < a1.Length; i++)
9.     {
10.        if (a1[i] != a2[i])
11.        {
12.            return false;
13.        }
14.    }
15.    return true;
16. }
```

Если длины массивов не совпадут, метод завершится быстрее

Время выполнения метода линейно возрастает с ростом длины общего префикса у a1 и a2

Timing-атаки

- **LAN:** разница в 200 наносекунд за 1000 измерений,
- **Internet:** разница в 30 микросекунд за 1000 измерений:

<http://www.cs.rice.edu/~dwallach/pub/crosby-timing2009.pdf>

Timing-атаки

Side-channel amplification: всегда есть
ВОЗМОЖНОСТЬ усилить канал.

<https://xaker.ru/2015/06/03/web-app-hack-keep-alive>

Добавление случайных временных задержек —
не поможет!

https://events.ccc.de/congress/2012/Fahrplan/attachments/2235_29c3-schinzl.pdf

Попытка #2 (.NET BCL-alike)

```
1. public static bool Equals(byte[] a1, byte[] a2)
2. {
3.     var result = true;
4.     for (var i = 0; i < a1.Length && i < a2.Length; ++i)
5.     {
6.         if (a1[i] != a2[i])
7.         {
8.             result = false;
9.         }
10.    }
11.    return result;
12.}
```



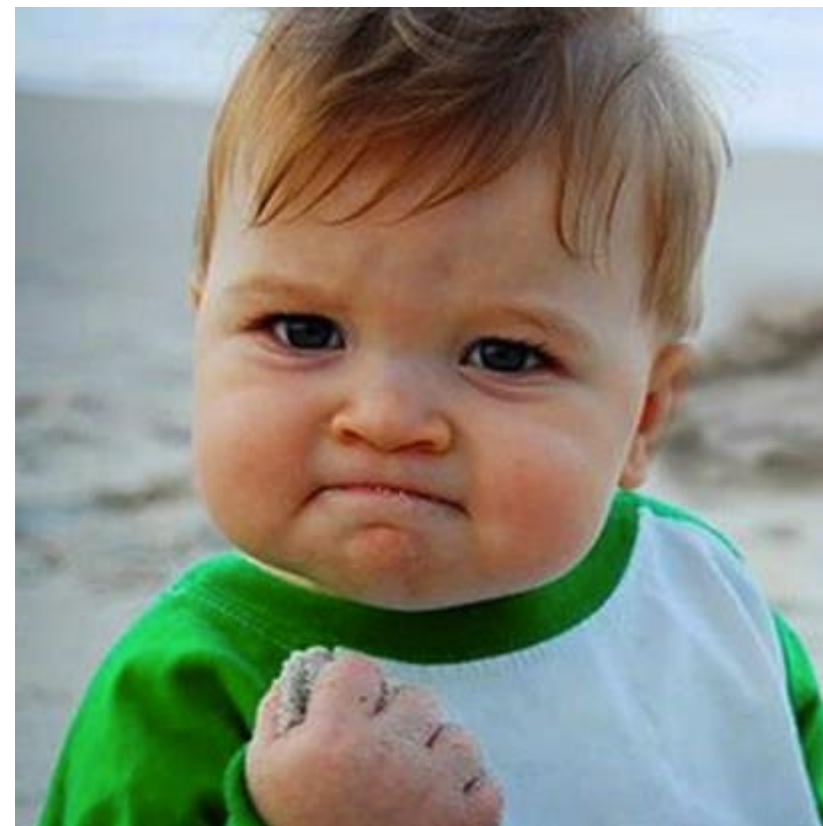
Попытка #2 (.NET BCL-alike)

```
1. public static bool Equals(byte[] a1, byte[] a2)
2. {
3.     var result = true;
4.     for (var i = 0; i < a1.Length && i < a2.Length; ++i)
5.     {
6.         if (a1[i] != a2[i])
7.         {
8.             result = false;
9.         }
10.    }
11.    return result;
12. }
```

Нет гарантии вмешательства в поток
выполнения оптимизаторов
языкового или JIT компиляторов

Правильный подход

```
1. public static bool Equals(byte[] a1, byte[] a2)
2. {
3.     var diff = (uint)a.Length ^ (uint)b.Length;
4.     for (var i = 0; i < a1.Length && i < a2.Length; i++)
5.     {
6.         diff |= (uint)(a1[i] ^ a2[i]);
7.     }
8.     return diff == 0;
9. }
```

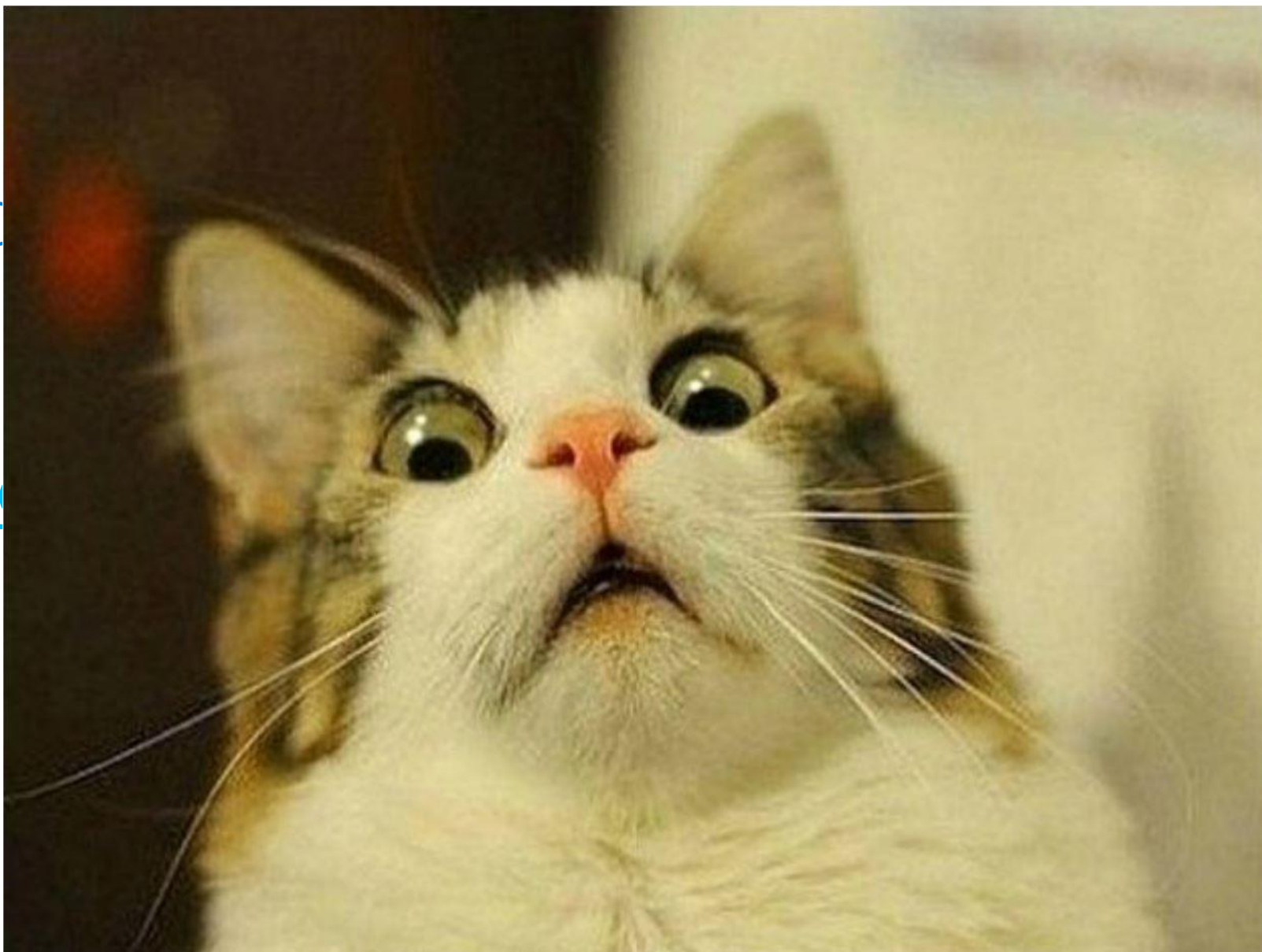


Как насчет выполнения **еще** свыше десятка
других правил?

https://cryptocoding.net/index.php/Coding_rules

Как нас
других

<https://c>



ка

[rules](https://c)

Годный план

- Для защищённой передачи данных использовать TLS
- HL-криптографию для всего остального:
 - Sodium (<https://libsodium.org> – биндинги для ~40 языков);
 - Keyczar (<https://github.com/google/keyczar> – C/C++, Java, Python, .NET);
 - Inferno (<https://github.com/sdrapkin/SecurityDriven.Inferno> – .NET);
 - Cryptography.io (<https://cryptography.io> – Python);
 - Halite (<https://paragonie.com/project/halite> – PHP);
 - ... (тысячи их)
- Средства низкоуровневой криптографии следует использовать только в исключительных и крайних случаях!

Нельзя просто так взять и использовать
низкоуровневую криптографию



Задача

Обеспечить конфиденциальность и целостность
передаваемых данных

Решение

Рассчитывать подпись по формуле

$$\text{signature} = \text{HASH}(\text{secret} \text{ — encrypted-data})$$

и передавать ее вместе с зашифрованными данными

Низкоуровневая криптография

```
1. private static bool IsValidSignature(string data, string signature)
2. {
3.     var bytes = Encoding.ASCII.GetBytes("eCTR4rhYQVNwn78j" + data);
4.     var hash = MD5.Create().ComputeHash(bytes);
5.     return BitConverter.ToString(hash) == signature;
6. }

7. ...

8. if (IsValidSignature(Request["data"], Request["signature"]))
9. {
10.    var decryptor = Aes.Create()
11.    {
12.        BlockSize = 128;
13.        Key = Encoding.ASCII.GetBytes("YtGDn6mvAHbp5X7C");
14.        IV = Encoding.ASCII.GetBytes("mHMUYSjiVxo4wp9R");
15.    }.CreateDecryptor();
16. }
```


Низкоуровневая криптография

```
1. private static bool IsValidSignature(string data, string signature)
2. {
3.     var bytes = Encoding.ASCII.GetBytes("eCTR4rhYQVNwn78j" + data);
4.     var hash = MD5.Create().ComputeHash(bytes);
5.     return BitConverter.ToString(hash) == signature;
6. }
7. ...
```

Жестко
заданный в коде
секрет

```
8. if (IsValidSignature(Request["data"], Request["signature"]))
9. {
10.    var decryptor = Aes.Create()
11.    {
12.        BlockSize = 128;
13.        Key = Encoding.ASCII.GetBytes("YtGDn6mvAHbp5X7C");
14.        IV = Encoding.ASCII.GetBytes("mHmUYSjiVxo4wp9R");
15.    }.CreateDecryptor();
16. }
```

Низкоуровневая криптография

```
1. private static bool IsValidSignature(string data, string signature)
2. {
3.     var bytes = Encoding.ASCII.GetBytes("eCTR4rhYQVNwn78j" + data);
4.     var hash = MD5.Create().ComputeHash(bytes);
5.     return BitConverter.ToString(hash) == signature;
6. }
7. ...
```

Уязвимость к атакам
удлинения сообщения

```
8. if (IsValidSignature(Request["data"], Request["signature"]))
9. {
10.    var decryptor = Aes.Create()
11.    {
12.        BlockSize = 128;
13.        Key = Encoding.ASCII.GetBytes("YtGDn6mvAHbp5X7C");
14.        IV = Encoding.ASCII.GetBytes("mHMUYSjiVxo4wp9R");
15.    }.CreateDecryptor();
16. }
```

Низкоуровневая криптография

```
1. private static bool IsValidSignature(string data, string signature)
2. {
3.     var bytes = Encoding.ASCII.GetBytes("eCTR4rhYQVNwn78j" + data);
4.     var hash = MD5.Create().ComputeHash(bytes);
5.     return BitConverter.ToString(hash) == signature;
6. }
```

Кто сказал, что
data будет в ASCII?

7. ...

```
8. if (IsValidSignature(Request["data"], Request["signature"]))
9. {
10.     var decryptor = Aes.Create()
11.     {
12.         BlockSize = 128;
13.         Key = Encoding.ASCII.GetBytes("YtGDn6mvAHbp5X7C");
14.         IV = Encoding.ASCII.GetBytes("mHMuYSjiVxo4wp9R");
15.     }.CreateDecryptor();
16. }
```

Низкоуровневая криптография

```
1. private static bool IsValidSignature(string data, string signature)
2. {
3.     var bytes = Encoding.ASCII.GetBytes("eCTR4rhYQVNwn78j" + data);
4.     var hash = MD5.Create().ComputeHash(bytes);
5.     return BitConverter.ToString(hash) == signature;
6. }
```

Использование
уязвимой функции
хэширования

7. ...

```
8. if (IsValidSignature(Request["data"], Request["signature"]))
9. {
10.     var decryptor = Aes.Create()
11.     {
12.         BlockSize = 128;
13.         Key = Encoding.ASCII.GetBytes("YtGDn6mvAHbp5X7C");
14.         IV = Encoding.ASCII.GetBytes("mHmUYSjiVxo4wp9R");
15.     }.CreateDecryptor();
16. }
```

Низкоуровневая криптография

```
1. private static bool IsValidSignature(string data, string signature)
2. {
3.     var bytes = Encoding.ASCII.GetBytes("eCTR4rhYQVNwn78j" + data);
4.     var hash = MD5.Create().ComputeHash(bytes);
5.     return BitConverter.ToString(hash) == signature;
6. }
```

Несбалансированное
сравнение строк

7. ...

```
8. if (IsValidSignature(Request["data"], Request["signature"]))
9. {
10.     var decryptor = Aes.Create()
11.     {
12.         BlockSize = 128;
13.         Key = Encoding.ASCII.GetBytes("YtGDn6mvAHbp5X7C");
14.         IV = Encoding.ASCII.GetBytes("mHmUYSjiVxo4wp9R");
15.     }.CreateDecryptor();
16. }
```

Низкоуровневая криптография

```
1. private static bool IsValidSignature(string data, string signature)
2. {
3.     var bytes = Encoding.ASCII.GetBytes("eCTR4rhYQVNwn78j" + data);
4.     var hash = MD5.Create().ComputeHash(bytes);
5.     return BitConverter.ToString(hash) == signature;
6. }
```

7. ...

```
8. if (IsValidSignature(Request["data"], Request["signature"]))
9. {
10.     var decryptor = Aes.Create()
11.     {
12.         BlockSize = 128;
13.         Key = Encoding.ASCII.GetBytes("YtGDn6mvAHbp5X7C");
14.         IV = Encoding.ASCII.GetBytes("mHMUYSjiVxo4wp9R");
15.     }.CreateDecryptor();
16. }
```

AES-CBC-PKCS7 →
уязвимость к атакам на
оракул дополнения

Низкоуровневая криптография

```
1. private static bool IsValidSignature(string data, string signature)
2. {
3.     var bytes = Encoding.ASCII.GetBytes("eCTR4rhYQVNwn78j" + data);
4.     var hash = MD5.Create().ComputeHash(bytes);
5.     return BitConverter.ToString(hash) == signature;
6. }
```

7. ...

```
8. if (IsValidSignature(Request["data"], Request["signature"]))
9. {
10.     var decryptor = Aes.Create()
11.     {
12.         BlockSize = 128;
13.         Key = Encoding.ASCII.GetBytes("YtGDn6mvAHbp5X7C");
14.         IV = Encoding.ASCII.GetBytes("mHMUYSjiVxo4wp9R");
15.     }.CreateDecryptor();
16. }
```

Жестко заданный в
коде ключ

Низкоуровневая криптография

```
1. private static bool IsValidSignature(string data, string signature)
2. {
3.     var bytes = Encoding.ASCII.GetBytes("eCTR4rhYQVNwn78j" + data);
4.     var hash = MD5.Create().ComputeHash(bytes);
5.     return BitConverter.ToString(hash) == signature;
6. }
```

7. ...

```
8. if (IsValidSignature(Request["data"], Request["signature"]))
9. {
10.     var decryptor = Aes.Create()
11.     {
12.         BlockSize = 128;
13.         Key = Encoding.ASCII.GetBytes("YtGDn6mvAHbp5X7C");
14.         IV = Encoding.ASCII.GetBytes("mHMUYSjiVxo4wp9R");
15.     }.CreateDecryptor();
16. }
```

Жестко заданный в
коде вектор
инициализации

Высокоуровневая криптография (почувствуйте
разницу)

Высокоуровневая криптография

```
1. // Keyczar (C#)
2. using(var crypter = new Crypter(keySet))
3. {
4.     decryptedData = crypter.Decrypt(ciphertext);
5. }
```

Высокоуровневая криптография

```
1. // Keyczar (C#)
2. using(var crypter = new Crypter(keySet))
3. {
4.     decryptedData = crypter.Decrypt(ciphertext);
5. }

6. // Sodium (Java)
7. SecretBox secretBox = new SecretBox(key);
8. decryptedData = secretBox.decrypt(nonce, ciphertext);
```

Высокоуровневая криптография

```
1. // Keyczar (C#)
2. using(var crypter = new Crypter(keySet))
3. {
4.     decryptedData = crypter.Decrypt(ciphertext);
5. }

6. // Sodium (Java)
7. SecretBox secretBox = new SecretBox(key);
8. decryptedData = secretBox.decrypt(nonce, ciphertext);

9. // Halite (PHP)
10.$decryptedData = Symmetric::decrypt($ciphertext, $key);
```

Высокоуровневая криптография

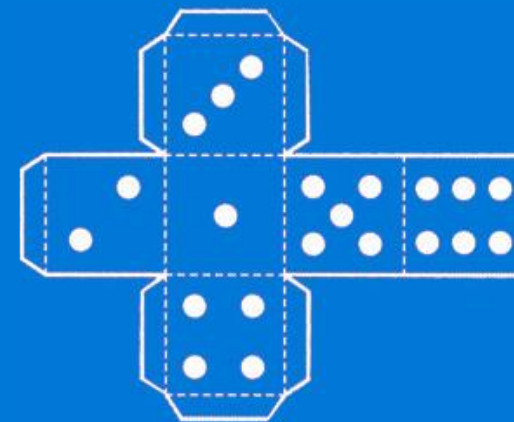
```
1. // Keyczar (C#)
2. using(var crypter = new Crypter
3. {
4.     decryptedData = crypter.De
5. }
```

```
6. // Sodium (Java)
7. SecretBox secretBox = new Secr
8. decryptedData = secretBox.decr
```

```
9. // Halite (PHP)
10.$decryptedData = Symmetric::de
```



Псевдослучайные числа



#msdevcon

Типовые сценарии использования

- Генерация:
 - параметров криптографических функций (векторов инициализации, okazji, значений соли, больших простых чисел и т.п.);
 - идентификаторов (фрагментов файловых путей, URL);
 - примитивов аутентификации (сессионных маркеров, временных паролей).
- Выбор ветви в рабочем потоке приложения.
- Специфика логики предметной области

А нужен ли вообще криптографический генератор псевдослучайных чисел?



Прямая и обратная защищённость PRNG

Что будет, когда атакующий воспроизведет **предыдущие** или **последующие** значения псевдослучайной последовательности?

Вопрос, сможет ли он это сделать в случае незащищённого PRGN – вообще не стоит.

Внезапно

GUID не предназначен для генерации случайных чисел

```
57fc4896-b324-45fe-b0f4-9ee200cdac16
81757edd-4605-49e7-bf13-9ee200cdac20
66605e1c-72ea-414c-a81e-9ee200cdac22
6836dc5c-f252-48af-b6fa-9ee200cdac23
dc56fed1-803c-494b-8f88-9ee200cdac24
18946c23-5111-4401-8f19-9ee200cdac25
cdacecd6-1337-4613-8dd7-9ee200cdac26
07335619-02ae-446f-b23d-9ee200cdac27
f37c14bd-4cda-4c21-8011-9ee200cdac28
0f8937d2-caa5-4c86-95eb-9ee200cdac29
9326688b-5822-49ba-8959-9ee200cdac2b
2bd7df82-fd77-4417-8533-9ee200cdac2c
282e0411-1789-4c43-af1e-9ee200cdac2d
e45c1c65-3b00-4ee5-bb97-9ee200cdac2e
06b89207-f06e-4f55-97a5-9ee200cdac2f
4cbb0eb9-998d-44e3-9b73-9ee200cdac30
209ab5b6-0d6d-4cc8-a78a-9ee200cdac31
4bda2caa-e69d-4c0b-8bff-9ee200cdac33
c126df6a-9374-469f-af84-9ee200cdac34
3a623042-74fc-46d6-94c3-9ee200cdac35
96a6e43e-73cb-4583-8200-9ee200cdac36
2f2e05e5-ef36-4d7e-b558-9ee200cdac37
de482a58-a592-4316-9a98-9ee200cdac38
```

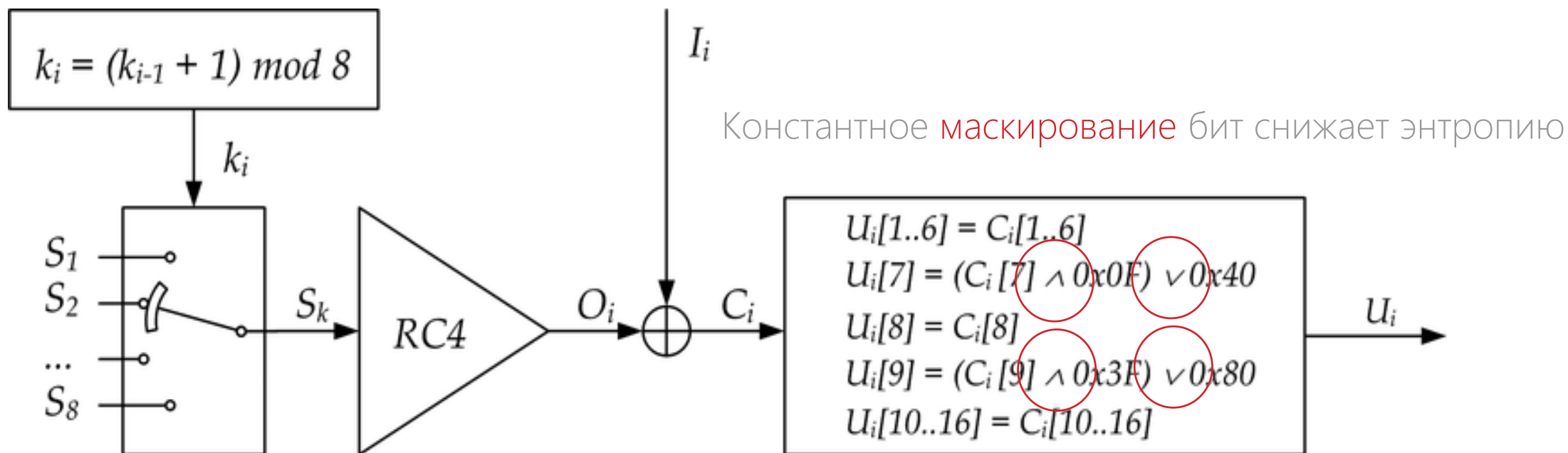
System.Guid (.NET Framework)

GUID предназначен для генерации уникальных идентификаторов → не гарантирует случайность генерируемой последовательности.

Обертка над **CoCreateGuid** (ole32.dll) → вызов **UuidCreate** (rpcrt4.dll), основанный на RC4 и уязвимый к атакам на угадывание очередного значения в ряде сценариев

System.Guid (.NET Framework)

<https://rstdn.ru/article/Crypto/UuidCrypto.xml>



Seed-гонка

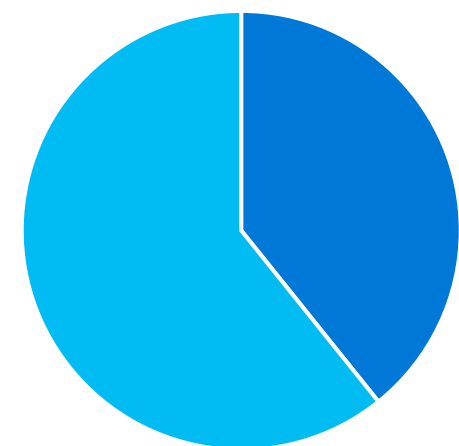
Состояние приложения, при котором PRNG, создаваемые в различных потоках, получают одно и то же значение seed.

Seed-гонка

Эксперимент **Neohapsis**

<http://labs.neohapsis.com/tag/seed-racing-attack-vector>

- 67 тысяч запросов на восстановление пароля;
- временный пароль генерировался с помощью System.Random (.Net Framework);
- получено 208 ответов с уникальными паролями;
- 322 ответа содержали одинаковые пароли.



■ Уникальные ■ Одинаковые

System.**Random** (.NET)

Значение seed по умолчанию:

Environment.**TickCount** → гонки за seed.

Субтрактивный LFG:

$$X_n = (X_{n-55} - X_{n-24}) \bmod (2^{31} - 1), n \geq 0$$

Для синхронизации с генератором достаточно получить последовательность из 55 случайных чисел.

System.**Random** (.NET)

Ошибка в реализации во всех версиях .NET
вплоть до текущей → неполный цикл
случайной последовательности.

$G(x) = x^{55} + x^{21} + 1$ ВМЕСТО $x^{55} + x^{24} + 1$

```
73         for (int k=1; k<5; k++) {
74             for (int i=1; i<56; i++) {
75                 SeedArray[i] -= SeedArray[1+(i+30)%55];
76                 if (SeedArray[i]<0) SeedArray[i]+=MBIG;
77             }
78         }
79         inext=0;
80         inextp = 21;
81         Seed = 1;
82     }
83
```

File: [system\random.cs](#)

Project: [ndp\clr\src\bcl\mscorlib.csproj](#) (mscorlib)

SSC.RandomNumberGenerator (.NET)

Абстрактный класс, в .NET BCL всего один наследник.

RNGCryptoServiceProvider – обертка над
CryptGetRandom (advapi32.dll) → FIPS-186-2
G=SHA1 (до Windows Vista SP1) и AES-CTR (все
последующие версии);

SSC.**RandomNumberGenerator** (.NET)

Не требуется задавать seed для генератора Crypto API.

Опасный метод **GetNonZeroBytes** снижает энтропию последовательности и не должен использоваться без необходимости.

java.util.**Random** и Math.**Random** (Java)

Вычисление seed:

$$\begin{aligned} \text{seed}_0 &= (\text{SU}++) + \text{current_timestamp}_{\text{nano}} \\ (\text{SU}_0 &= 8682522807148012) \\ \text{seed}_{i+1} &= (\text{seed}_0 ^{25214903917}) \& ((1L \ll 48) - 1) \end{aligned}$$

Math.**Random** – обёртка над java.util.**Random**

Math.Random и java.util.Random (Java)

Классический LCG:

$$X_n = (25214903917 * X_{n-1} + 11) \bmod 2^{48}, n \geq 0$$

Для синхронизации с генератором достаточно получить последовательность из 2 случайных чисел.

java.security.**SecureRandom** (Java)

Инкапсулирует 4 различных PRNG:

Реализация	Платформа
sun.security.mscapi.PRNG	Windows
sun.security.provider.nativePRNG	*nix
sun.security.provider. SecureRandom	Не зависит от платформы
sun.security.provider.pkcs11.P11SecureRandom	N/A (PKCS)

java.security.**SecureRandom** (Java)

Защищённость

sun.security.provider.**SecureRandom** зависит от конкретной реализации генератора.

https://www.hgi.rub.de/media/nds/veroeffentlichungen/2013/03/25/paper_2.pdf

lcg_value() (PHP)

Вычисление seed:

$$\text{Seed}_1 = \text{timestamp} \wedge (\text{microseconds2} \ll 11)$$

$$\text{Seed}_2 = \text{pid} \wedge (\text{microseconds3} \ll 11)$$

microseconds2 на 0-3 больше, чем при первом замере (microseconds1)

pid — process id текущего процесса (0-32768)

microseconds3 на 1-4 больше microseconds2

lcg_value() (PHP)

Комбинированный MLCG:

MLCG1:

$$s_{n+1}^1 = (s_n^1 * a_1) \bmod m_1 = (s_n^1 * 40014) \bmod (2^{31} - 85)$$

MLCG2:

$$s_{n+1}^2 = (s_n^2 * a_2) \bmod m_2 = (s_n^2 * 40692) \bmod (2^{31} - 249)$$

Combined MLCGs:

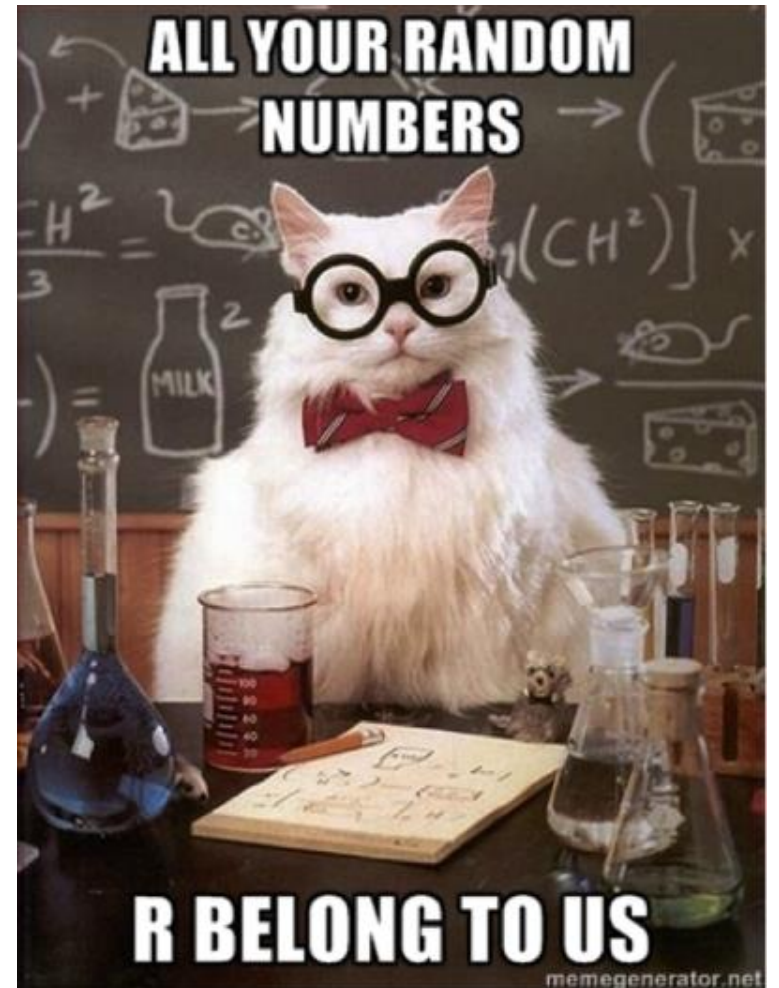
$$z_{n+1} = (s_{n+1}^1 - s_{n+1}^2) \bmod (m_1 - 1) = (s_{n+1}^1 - s_{n+1}^2) \bmod (2^{31} - 84)$$

Для синхронизации с генератором достаточно получить последовательность из 3 случайных чисел.

`mt_rand()`, `rand()`, `uniqid()`, `shuffle()` (PHP)

Случайные числа: take two

<https://habrahabr.ru/company/pt/blog/149746>



Устойчивые генераторы (PHP)

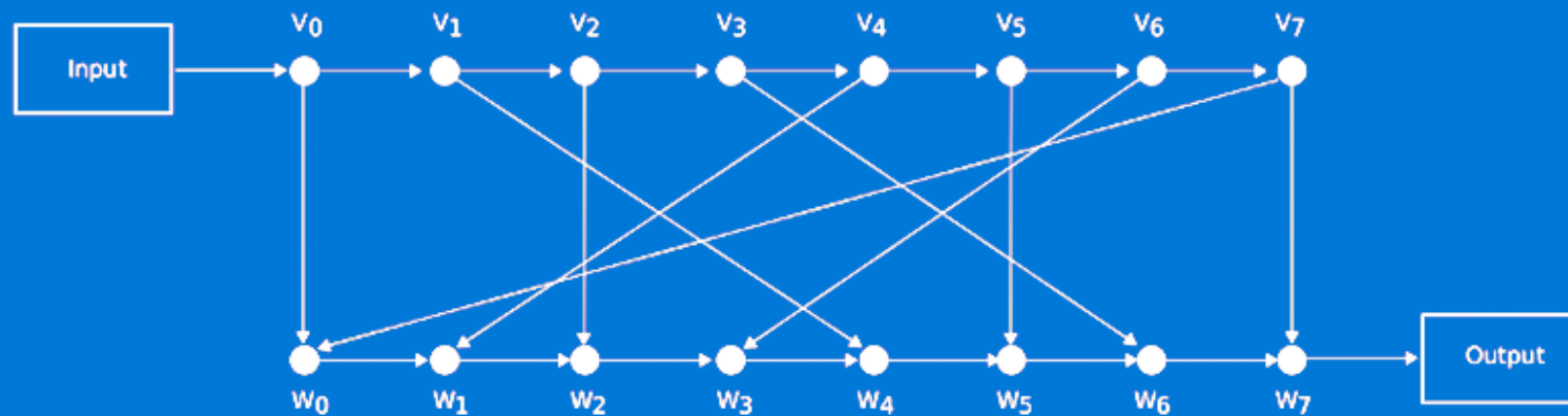
openssl_random_pseudo_bytes():

- Требует контроля за выходным аргументом `&$crypto_strong;`
- только PHP v5.6.10+ (в ранних версиях эта функция использовала уязвимый `RAND_pseudo_bytes` вместо `RAND_bytes`);
- проблемы random fork safety.

random_bytes():

- Windows → `CryptGetRandom()`, Linux → `getrandom(2)`, все остальные → `/dev/urandom`.

Хеширование



Типовые сценарии использования

- Генерация идентификаторов:
 - пользовательских паролей;
 - фрагментов URI;
 - внешних данных.
- Подтверждение аутентичности и целостности
- Построение криптографических примитивов

Хэширование (.NET)

- `System.Security.Cryptography.KeyedHashAlgorithm`:
 - семейство `HMAC*`: ОК (с учетом безопасности базовых хэш-функций);
 - `MACTripleDES`: только для обратной совместимости.
- семейство `SHA-2` (256/384/512): ОК (пока)

Хэширование (.NET)

- MD5 и SHA-1: не должны использоваться!
- RIPEMD160: только для обратной
совместимости

Хэширование (PHP, Java)

- PHP: MD2/4/5, SHA-1, SHA-2, RIPEMD, Whirpool, Tiger, Snefru, Gost, Gost-crypto, Adler, CRC, FNV, Joaat, Haval.
- Java: MD5, SHA-1, SHA-2.

Хэширование (PHP, Java)

- PHP: MD2/4/5, SHA-1, **SHA-2**, RIPEMD, Whirpool, Tiger, Snefru, Gost, **Gost-crypto**, Adler, CRC, FNV, Joaat, Haval.
- Java: MD5, SHA-1, **SHA-2**.

Задача

Обеспечить целостность передаваемых открытых данных

#msdevcon

Очевидное решение

Рассчитывать подпись по формуле

$$\text{signature} = \text{HASH}(\text{secret} \text{ — plaintext-data})$$

и передавать ее вместе с данными

Проблема

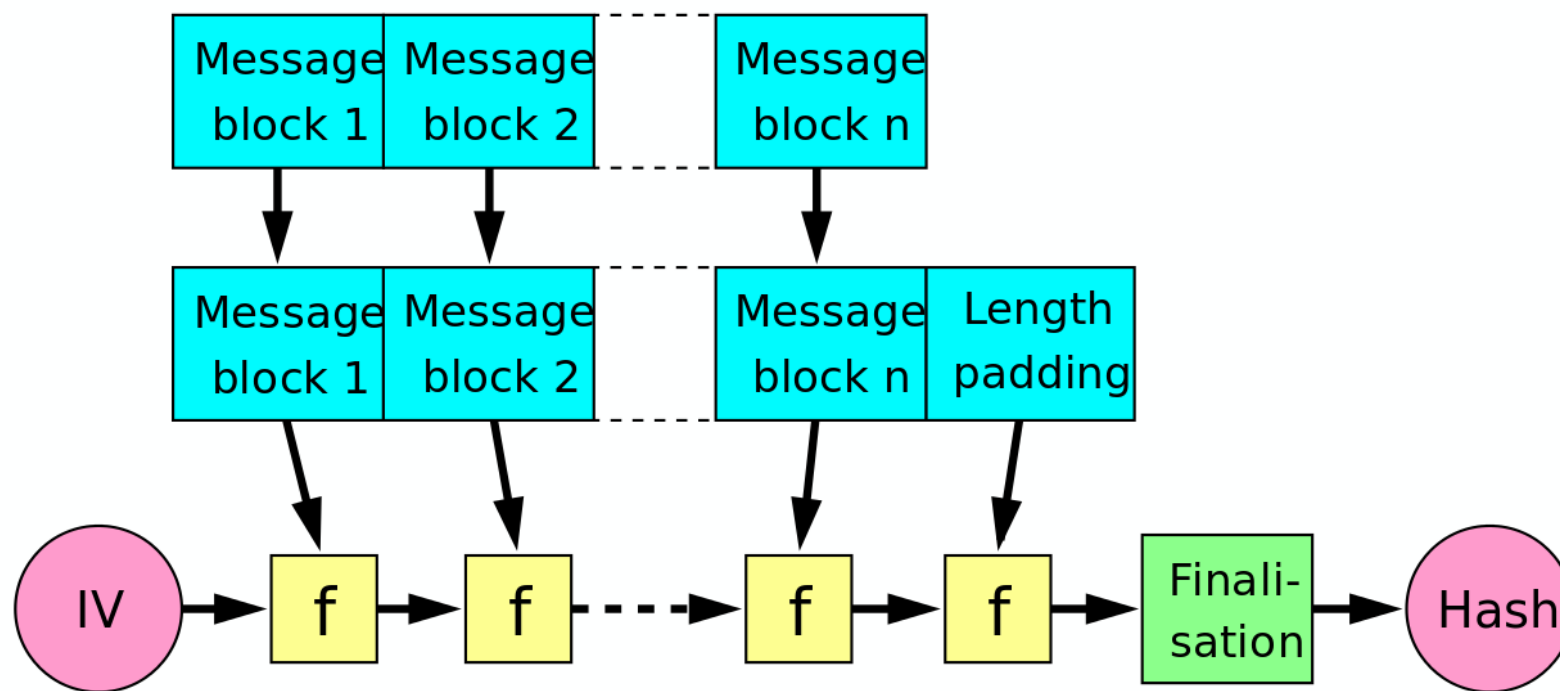
При таком решении атакующий сможет подделывать подпись для

user-data — attacker-data

генерируемую хэш-функциями на основе структуры Меркла-Дамгарда

Структура Меркла-Дамгарда

Лежит в основе хеш-функций: MD4, MD5, RIPEMD-160, SHA-0, SHA-1, SHA-2, WHIRPOOL



Атака удлинением сообщения

$$m' = \text{user-data} \text{ — user-padding}$$

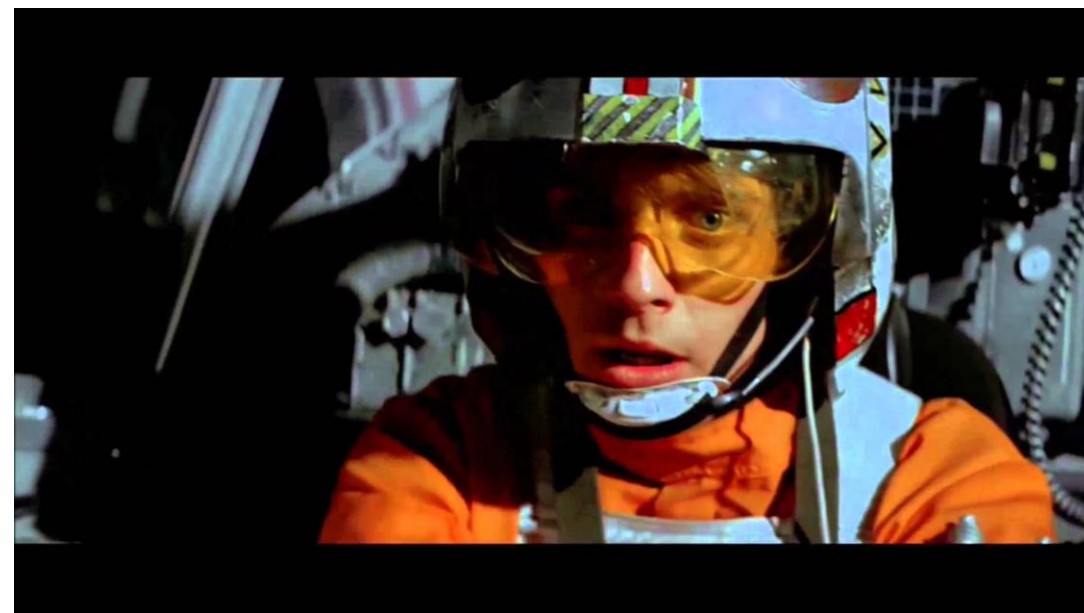
Атакующему достаточно вычислить хеш для:

$$m'' = \text{attacker-data} \text{ — attacker-padding}$$

инициализировав хеш-функцию значением m'

Как защититься?

Используй **НМАС**, Люк!



Как защититься?

Используй **HMAC**, Люк!

(ну, или в крайнем случае: $\text{hash}(\text{data} \text{ — } \text{secret})$ или $\text{hash}(\text{secret} \text{ — } \text{hash}(\text{secret} \text{ — } \text{data}))$)



Хранение паролей



#msdevcon

Общие принципы

Криптографические функции хеширования не подходят для задачи хранения учётных данных.

Для хэширования паролей следует использовать адаптивные функции **Argon2** (<https://password-hashing.net>), **PBKDF2**, **scrypt**, **bcrypt**:

$$\text{hash} = \text{salt} \frown \text{adaptive_hash}(\text{password}, \text{salt})$$

или дайджест-функции:

$$\text{hash} = \text{salt} \frown \text{HMAC-SHA-256}(\text{password}, \text{salt}, \text{secret-key})$$

Соль

Предназначение соли — затруднение атак по словарям и радужным таблицам.

Соль не является секретом и должна быть случайной и уникальной для каждого пароля.

Длина соли должна быть достаточной для обеспечения энтропии **salt** — **password** ≥ 256 бит для любого возможного пароля → длина соли ≥ 32 байта.

[https://www.owasp.org/index.php/Password Storage Cheat Sheet](https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet)

Хранение паролей (.NET)

```
1. public static string GetPasswordHash(string password)
2. {
3.     Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, 32);
4.     rfc2898DeriveBytes.IterationCount = 16384;
5.     byte[] hash = rfc2898DeriveBytes.GetBytes(32);
6.     byte[] salt = rfc2898DeriveBytes.Salt;
7.     return Convert.ToBase64String(salt) + "|" + Convert.ToBase64String(hash);
8. }
```

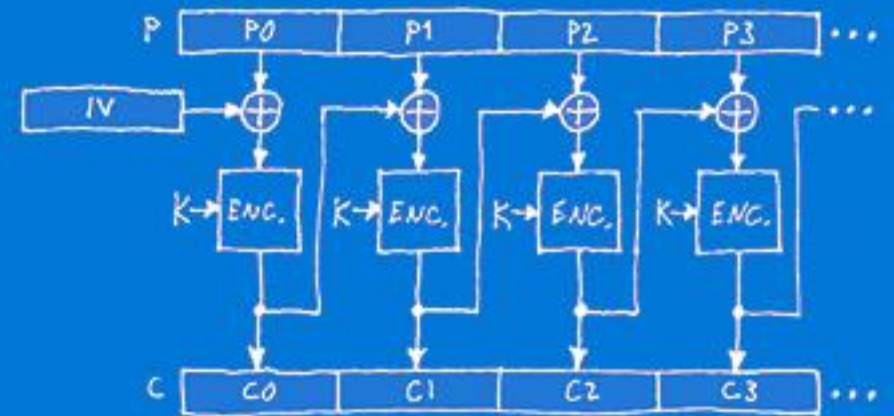
Хранение паролей (Java)

```
1. public static String getPasswordHash(final char[] password, final byte[] salt)
2. {
3.     try
4.     {
5.         SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA512");
6.         PBEKeySpec spec = new PBEKeySpec(password, salt, 16384, 32);
7.         SecretKey key = skf.generateSecret(spec);
8.         byte[] res = key.getEncoded();
9.         String saltString = new String(Base64.encodeBase64(salt));
10.        String saltRes = new String(Base64.encodeBase64(res));
11.        return saltString + "|" + saltRes;
12.    }
13.    catch(NoSuchAlgorithmException | InvalidKeySpecException e)
14.    {
15.        throw new RuntimeException(e);
16.    }
17.}
```

Хранение паролей (PHP)

```
1. <?php
2. function get_password_hash($password)
3. {
4.     return password_hash($password, PASSWORD_BCRYPT, ['cost' => 16384]);
5. }
```

Блочные шифры



Блочные шифры

AES: (блок 128 бит, ключ 128/192/256 бит)

Rijndael: (блок 128/192/256 бит, ключ 128/192/256 бит)

RC2 и TripleDES (DESede в Java): только для обратной
совместимости

DES: не должен использоваться!

Все прочие – только при объективной
необходимости

Критичные параметры блочных шифров

IV – вектор инициализации:

- **должен** генерироваться случайным образом;
- **не должен** использоваться повторно;
- **может** не являться секретом.

Key – ключ шифрования:

- **должен** генерироваться случайным образом (см. RFC 2898);
- **не должен** использоваться повторно;
- **должен** являться секретом.

Критичные параметры блочных шифров

Cipher mode - режим шифрования

ECB:

только в качестве базы для неподдерживаемых режимов!

CBC:

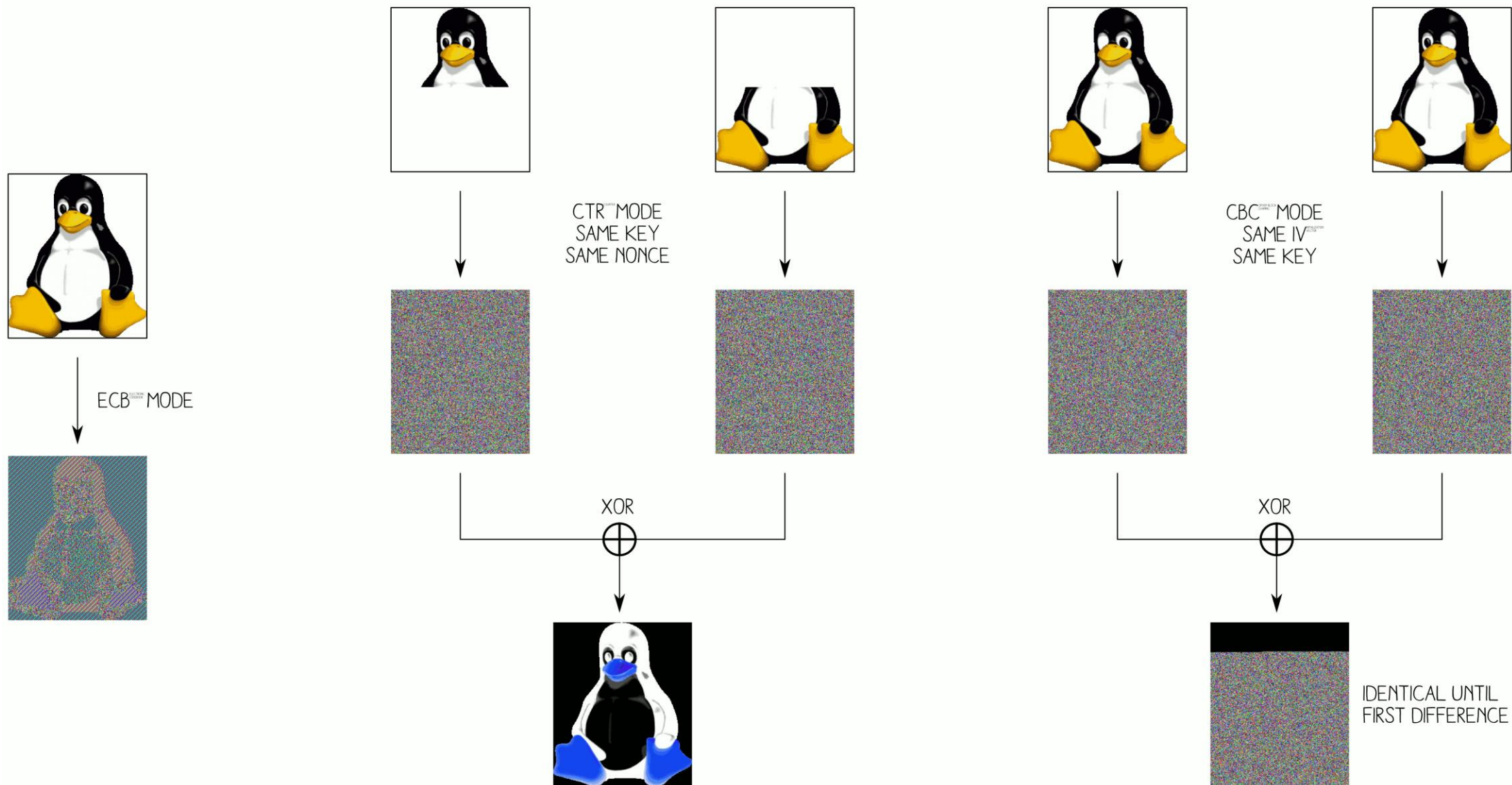
ОК, при условии дополнительной реализации EtA.

CFB, OFB, CTR:

обеспечивают конфиденциальность шифротекста, основаны на IV, имитируют потоковое шифрование, часто допускают (хотя и не требуют) дополнение.

Modes of operation's failures

ANGE ALBERTINI - CORKAMI.COM
AND THE HELP OF JEAN-PHILIPPE AUMASSON
VERSION 1.00
2014/01/21



Критичные параметры блочных шифров

Padding Mode - режим дополнения блоков:

ANSIX923: [FF FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 07]

ISO10126: [FF FF FF FF FF FF FF FF FF FF 7D 2A 75 EF F8 EF 07]

None: [FF FF FF FF FF FF FF FF FF FF]

PKCS7: [FF FF FF FF FF FF FF FF FF FF 07 07 07 07 07 07 07]

Zeros: [FF FF FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00]

Проблема

В случае использования **AES-CBC-PKCS7**, если атакующий может модифицировать шифротекст и отслеживать возникающие ошибки удаления дополнения...

... то он сможет **расшифровать** и **зашифровать*** произвольное сообщение, не обладая при этом ключом!

*за исключением первого блока

 Демонстрация

Атака на оракул дополнения

#msdevcon

Уязвима не только конфигурация **AES-CBC-
PKCS7!!!**

Атаки на оракул дополнения

Уязвим **любой** блочный шифр

Уязвимы **все** режимы дополнения, кроме **ISO10126** и **None**

{Частично} уязвимы **все** доступные режимы конфиденциального шифрования, допускающие дополнение и устанавливающие обратную связь по шифротексту или ключевому потоку между всеми блоками (**CBC, OFB, CFB, CTR, ...**)

Перехват исключений и добавление временных задержек **не устраняют** уязвимость

Как защититься?

Использовать средства
высокоуровневой криптографии

Использовать режимы
аутентичного шифрования (**GCM**,
CWC, **EAX**, **CCM**, **OCB**)

Реализовать EtA вручную, поверх
используемого режима



Подход EtA (encrypt-then-authenticate)

- Сообщение шифруется обычным образом (например, **AES-CBC**)
- **IV** добавляется к зашифрованному сообщению
- Для полученного на шаге 2 считается MAC с помощью одной из реализаций **HMAC** (например, **HMACSHA512**) и также добавляется к нему
- ...
- PROFIT!!! (и никаких оракулов)

Полезняшки

Разбор и анализ реализаций PRNG в стандартных билблитеках PHP, Java, .NET, C/C++
(http://portal.idc.ac.il/en/schools/cs/research/documents/sinai_2011.pdf)

Серия статей о техниках взлома различных PRNG (<https://jazzy.id.au/tags/prng.html>)

Рекомендации по генерированию SPRNG во многих языках
(<https://paragonie.com/blog/2016/05/how-generate-secure-random-numbers-in-various-programming-languages>)

Подробный разбор атак удлинением сообщения
(<https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks>) и оракула дополнения (<https://blog.skullsecurity.org/2013/padding-oracle-attacks-in-depth>)

 Что дальше

Круглый стол

Приходите вечером на
круглый стол.
Обсудим в узком круге.

Обратная связь

Присылайте отзывы и
вопросы (контакты в
следующем слайде)

Часть II

15.12.2016 14:00

<https://www.ptsecurity.com/ru-ru/research/webinar/>

(регистрация открывается за
неделю до вебинара)

#msdevcon



Q&A

Практическое использование средств криптографии в .NET, Java и PHP

Владимир Кочетков

<https://kochetkov.github.io/>

vkochetkov@ptsecurity.com

#msdevcon

Помогите нам стать лучше!

На вашу почту отправлена индивидуальная ссылка на электронную анкету. 2 ноября в 23:30 незаполненная анкета превратится в тыкву.

Заполните анкету и подходите к стойке регистрации за приятным сюрпризом!

#msdevcon

Оставляйте отзывы в социальных сетях. Мы все читаем. Спасибо вам! 😊

