Криптография с закрытым ключом

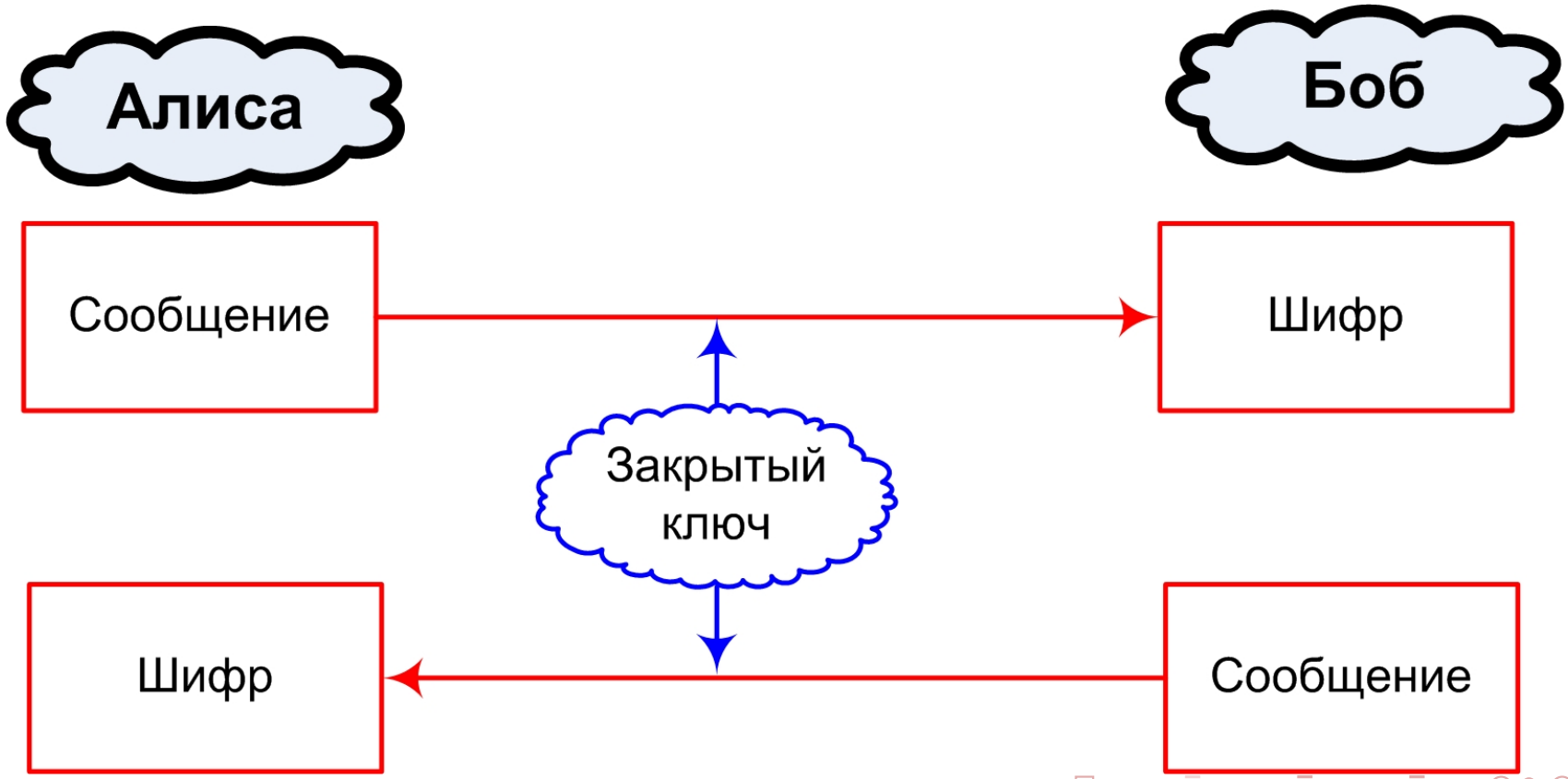
Сергей Николенко

Computer Science Club, 2015

Outline

Суть происходящего

- Алиса и Боб имеют общий секретный ключ.
- Они могут передавать сообщения:



Одноразовые блокноты

- Несложно сделать абсолютно непробиваемый шифр — *одноразовый блокнот*.
- Алиса и Боб суммируют ключ с сообщением побитово:

$$c = m \oplus k_{AB}.$$

- У врага нет никаких шансов дешифровать такое сообщение.

Задача

- Проблема: мы не можем себе позволить так относиться к ключам.
- Надо использовать один и тот же ключ много раз.
- Поточное кодирование: идёт поток данных, нужно кодировать каждый новый бит и передавать дальше в поток.
- Блочное кодирование: ключ один (не слишком длинный), сообщение гораздо длиннее, его приходится разбивать на блоки и кодировать блоки.
- Но сначала поговорим о самих шифрах.

Подстановочные шифры

- Подстановочные шифры (substitution ciphers): пусть $e \in S_A$ (e — перестановка букв алфавита). Тогда для e можно определить кодирующее преобразование:

$$E_e(m) = (e(m_1)e(m_2)\dots e(m_t)) = (c_1c_2\dots c_t) = c.$$

- Количество перестановок на достаточно большом алфавите велико (русский: $33! \approx 8,68 \cdot 10^{36}$).
- Но если просто так шифровать, будет очень плохо (почему?).

Гомофонные подстановочные шифры

- Homophonic substitution ciphers.
- То же самое, но теперь не перестановка символ–символ, а функция $A \longrightarrow 2^{A^t}$.
- Например:

$$A = \{0, 1\}, \quad e(a) = \{00, 01\}, \quad e(b) = \{10, 11\}.$$

- Теперь можно кодировать частые символы разными кодами, уравнивая частоту встречаемости.

Полиалфавитные шифры

- Полиалфавитный подстановочный шифр: рассмотрим вектор перестановок $\mathbf{e} = (e_1, e_2, \dots e_t)$ и будем кодировать блок сообщения $m_1 m_2 \dots m_t$ как

$$E_{\mathbf{e}} = (e_1(m_1), e_2(m_2), \dots, e_t(m_t)).$$

- Шифр Виженера — из таких.
- Не сильно лучше, чем моноалфавитный: можно разбить сообщение на части, зашифрованные одной перестановкой, и криптоанализировать по отдельности.
- Нужно только определить период; для этого и был метод Касиски и прочие методы.

Перестановочные шифры

- Ещё можно не менять буквы в сообщении, а переставлять их.
- По отдельности это тоже легко поддаётся криптоанализу.
- Поэтому правильный подход — всё объединить.

Желаемое

- Предположим, я хочу зашифровать блок размером n битов.
- Конечно, хотелось бы просто указать в качестве ключа случайную перестановку всех возможных блоков.
- Какой тогда будет размер ключа?

Желаемое

- Предположим, я хочу зашифровать блок размером n битов.
- Конечно, хотелось бы просто указать в качестве ключа случайную перестановку всех возможных блоков.
- Какой тогда будет размер ключа?
- Всего ключей 2^n, перестановок — $2^n!$.
- Длина ключа — $\log(2^n!) \approx n2^n - \frac{1}{\ln 2}2^n + \frac{n}{2}$ битов (формула Стирлинга).

Действительное

- Случайную замену для всего блока (скажем, блок из 64 битов) указать трудно.
- Поэтому шифры делают из *замен* (substitutions) и *перестановок* (permutations).
- Шифр — это композиция более простых замен и перестановок.

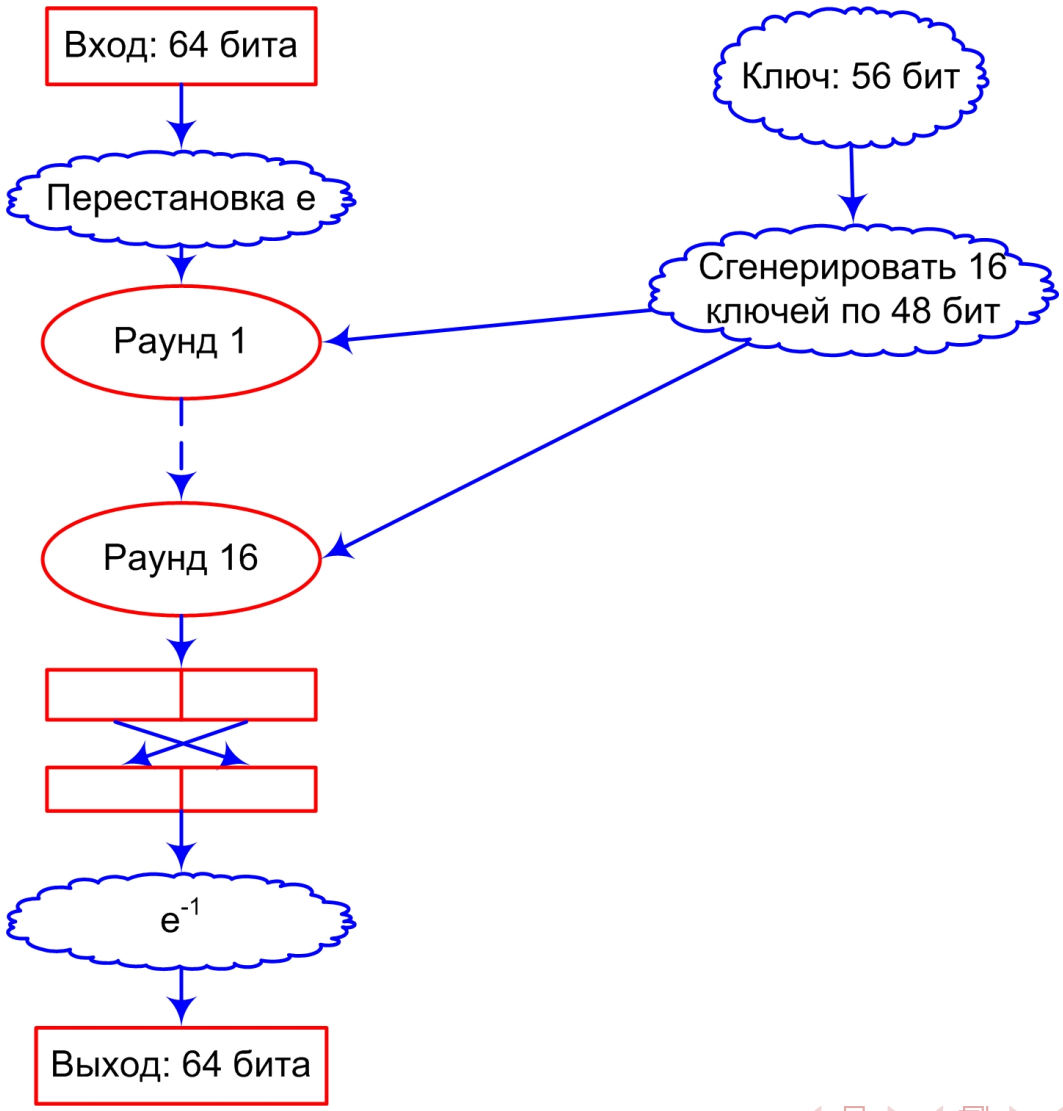
64-битный шифр

- Например, блоки длины 64 можно разбить на куски длины 8.
- Если ограничиться заменами и перестановками внутри кусков, то будет легко взломать.
- Поэтому в шифре несколько *раундов*; внутри раунда работа идёт внутри кусков по 8 бит, между раундами они перемешиваются.

Пример: DES

- DES (data encryption standard) — уже устарел, но его проще будет разобрать.
- Ключ длиной 56 битов используется для кодирования 64-битного блока.
- При помощи этого ключа генерируются 16 ключей для 16 раундов.

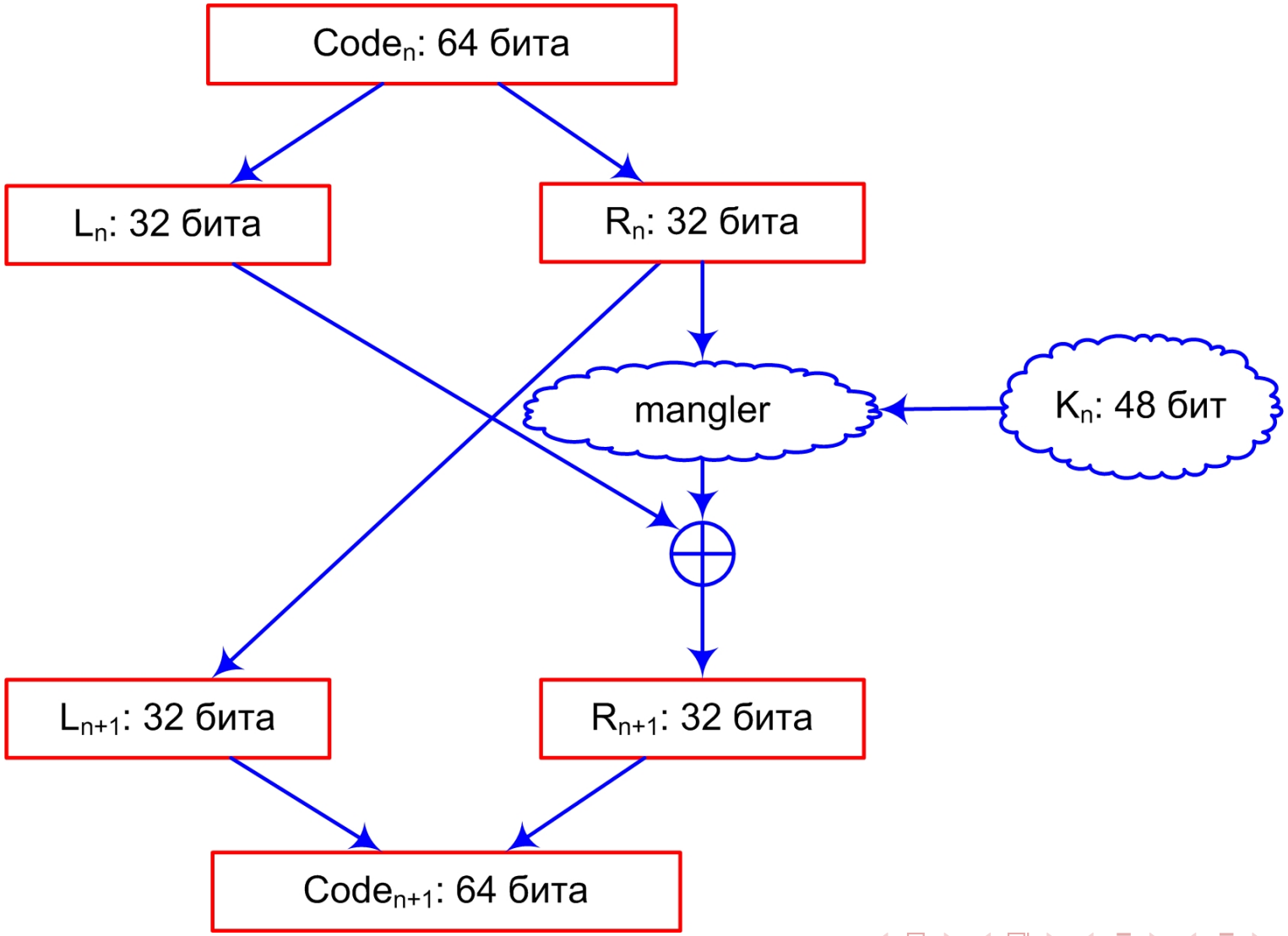
Пример: DES


Вход: 64 бита
Ключ: 56 бит
Перестановка e
Сгенерировать 16 ключей по 48 бит
Раунд 1
Раунд 16
e^{-1}
Выход: 64 бита

Пример: DES

- Перестановка e — по сути не нужна (зачем?).
- Генерация ключей — 16 раундов по перестановке, левая и правая половины ключа создаются отдельно.
- Каждый раунд — применение специальной mangler-функции к ключу и половине кода.

Пример: DES


Code$_n$: 64 бита
L$_n$: 32 бита
R$_n$: 32 бита
mangler
K$_n$: 48 бит
L$_{n+1}$: 32 бита
R$_{n+1}$: 32 бита
Code$_{n+1}$: 64 бита

История DES

- Был разработан в начале 1970-х, принят как стандарт в 1976.
- Разрабатывало IBM, но NSA (National Security Agency) принимало активное участие, меняло протоколы.
- Думали, что особенности DES позволяли NSA его дешифровывать.
- На самом деле функции выбирались так, чтобы их было труднее взламывать разностным криптоанализом (differential cryptoanalysis). Это метод для взлома блочных шифров; Eli Biham, Adi Shamir, конец 1980-х; но, видимо, IBM и NSA знали об этих атаках ещё в 1970-х (Don Coppersmith, 1994).

Современные альтернативы

- IDEA (International Data Encryption Algorithm), 1991 — блоки по 64 бита, ключ 128 битов. Чередует побитовый XOR, сложение и умножение по модулю 2^{16}.
- AES (Advanced Encryption Standard), 2001 — официально пришёл на смену DES. На основе семейства кодов Rijndael. Ключи могут быть размером 128, 192 и 256 битов.

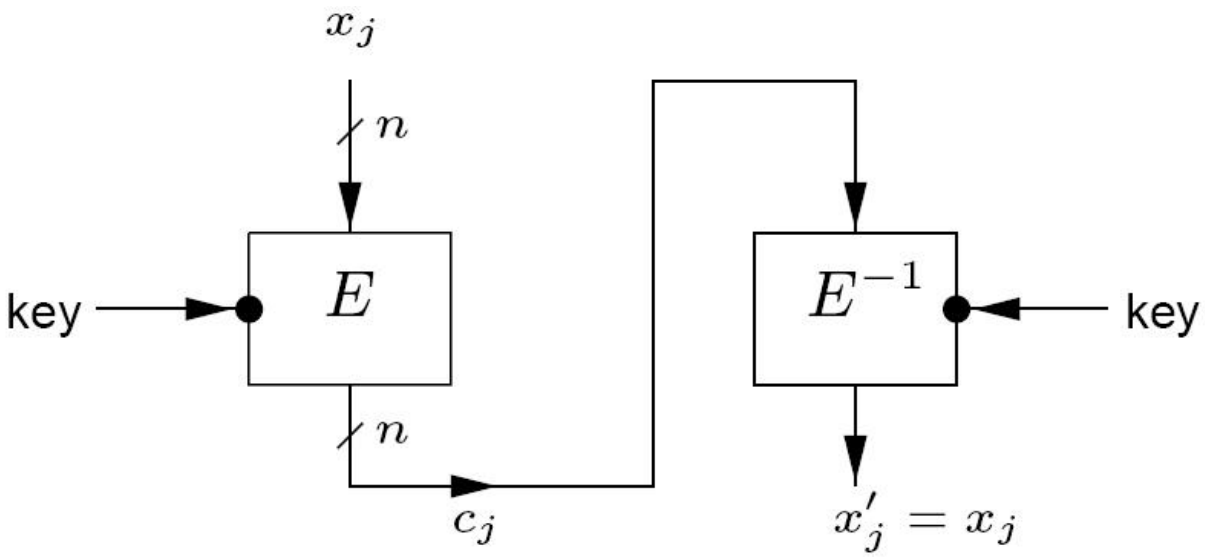
Outline

Постановка задачи

- Вернёмся теперь к блочным шифрам.
- Предположим, что у нас есть хороший метод кодировать блоки длиной 64 бита.
- Но сообщение — длиннее. Что делать?
- Мы сейчас будем рассматривать разные методы построения шифров для длинных сообщений из маленьких блоков.

ECB

- Самое простое - ECB (electronic codebook); блоки кодируются независимо друг от друга.
- $c_i = E(m_i)$, $m_i = D(c_i)$. Что плохо?



ECB

- Одинаковые блоки переходят в одинаковый код.
- Блоки кодируются независимо; следовательно, их можно переставлять.
- Ошибка в одном блоке дальше этого блока не идёт.
- Из-за первого и второго свойств атаковать можно очень успешно, и ECB использовать не рекомендуется.

Randomized ECB

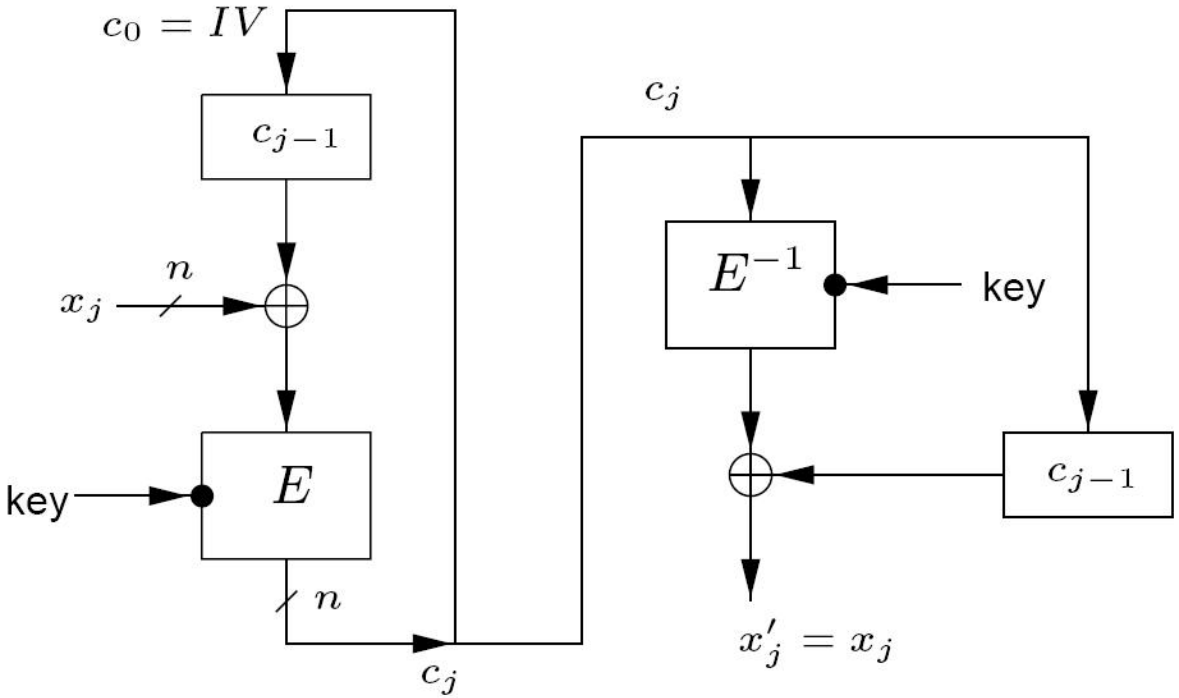
- OK, совсем плохо получилось, что одинаковые блоки кодируются одинаково. Давайте рандомизируем.
- Будем выбирать случайное число r_i для каждого блока, кодировать $c_i = E(m_i \oplus r_i)$ и передавать c_i и r_i.
- Теперь одинаковые m_i кодируются совершенно разными c_i.
- Что плохо?

Randomized ECB

- Во-первых, эффективность – фактически вдвое раздули сообщение.
- Во-вторых, по-прежнему можно переставлять блоки.
- И самое интересное – теперь можно, подменив r_i, вызывать предсказуемые изменения в дешифрованном m_i.

CBC

- CBC (cipher block chaining): $c_0 := IV$, $c_j := E_K(x_j \oplus c_{j-1})$; тогда $m_j = D(c_j) \oplus c_{j-1})$.
- Т.е. теперь «случайные числа» r_i зависят от предыдущих:



CBC

- Идентичные сообщения переходят в идентичные коды, только если IV тоже совпадает.
- Блок зависит от предыдущих, переставлять нельзя.
- Всё хорошо? Можно ли что-то сделать с таким сообщением?

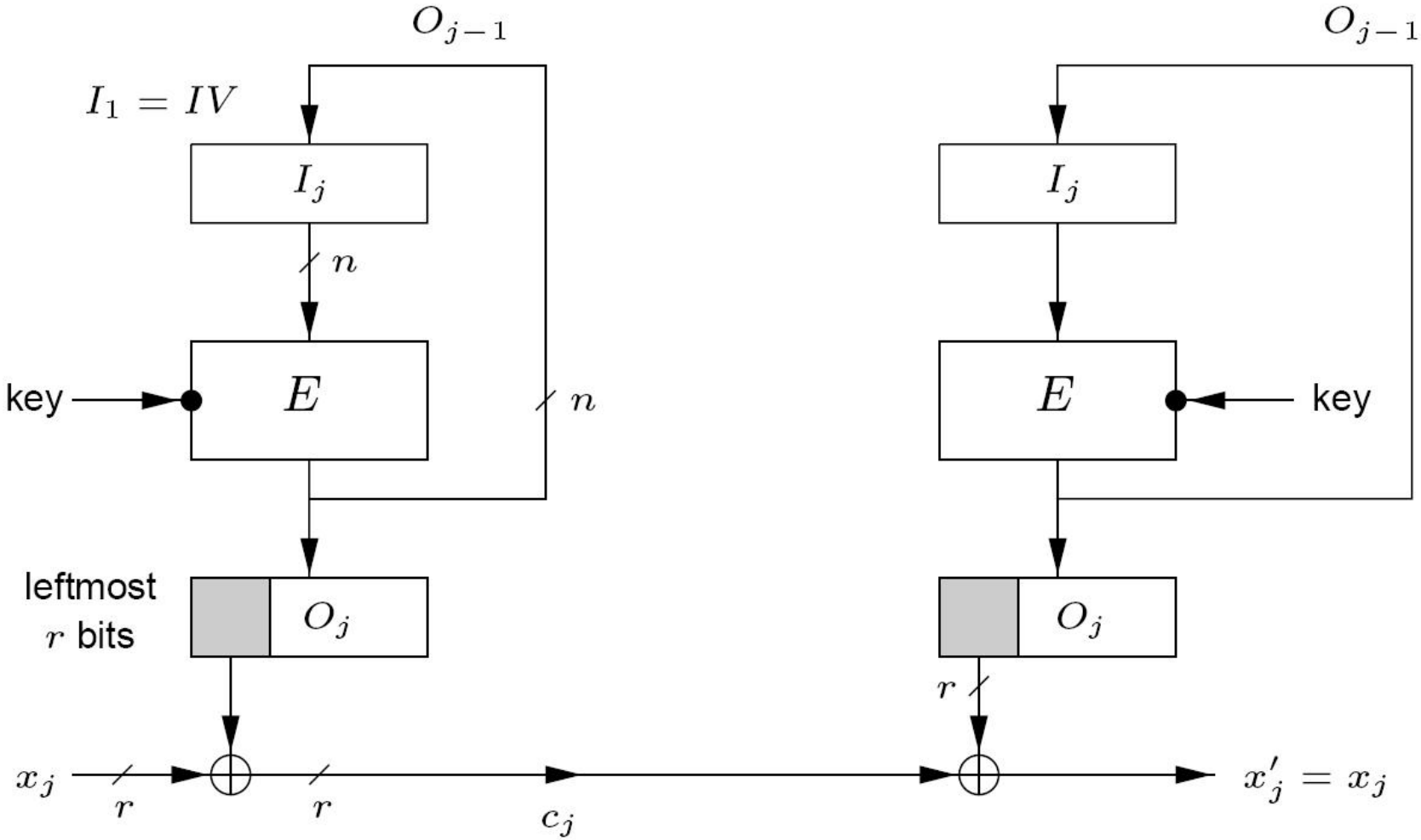
CBC

- Изменение в блоке c_j вызывает изменение в x_{j+1} на том же месте, дальше всё OK; но при этом x_j теряется полностью.
- Т.е. враг может подкорректировать x_{j+1}, но только ценой того, что x_j превратится в мусор.
- Ещё одна достаточно эзотерическая атака: предположим, что противник знает сообщение $m_1, \ldots, m_n$ и код $IV, c_1, \ldots, c_n$.
- Тогда противник может вычислить $D(c_j)$, а затем переставлять c_i и декодировать переставленные сообщения.
- Но при таком подходе, конечно, должно противнику сильно повезти, чтобы получилось что-то разумное и ему полезное в результате.

OFB

- OFB (output feedback) – это фактически потоковый шифр: мы кодируем, делая XOR с «одноразовым блокнотом», который мы вычисляем на каждом шаге.
  - берём $I_0 := IV$, $I_1 := E(I_0)$, ...
  - кодируем $c = m \oplus I$, где I – начало последовательности I_j нужной длины;
  - передаём c и I_0.

OFB


O_{j-1}
$I_1 = IV$
I_j
n
key
E
n
leftmost
r bits
O_j
x_j
r
r
c_j
O_{j-1}
I_j
E
key
O_j
r
$x'_j = x_j$

OFB

Преимущества:

- Одноразовый блокнот можно сгенерировать заранее, не зная сообщения. Тогда декодирование становится очень быстрым.
- Если в код закралась ошибка, теряются только соответствующие биты сообщения, остальные не изменяются (раньше мы теряли или целый блок, или два блока).
- Можно передавать кусками произвольного размера, продолжая с любого места.
- Идентичные сообщения переходят в идентичные коды, только если IV тоже совпадает.

OFB

- Недостатки:
  - Что может сделать противник, если он знает и сообщение, и его код?

OFB

- Недостатки:
  - Что может сделать противник, если он знает и сообщение, и его код?
  - Противник может изменить сообщение m_i на что угодно m_i', затем подсчитать код m_i' как $c_i \oplus m_i \oplus m_i'$.
  - Блок не зависит от предыдущих. Поэтому, в частности, нельзя использовать тот же ключ с тем же IV два раза.
  - Ошибка в блоке c_j к другим ошибкам не приводит, но зато потерялась самосинхронизация.

Вопрос

- В OFB, если мы хотим декодировать часть большого сообщения (скажем, начать читать файл не с начала), нам всё равно придётся начать декодировать сначала).
- Как модифицировать OFB, чтобы так не получалось?

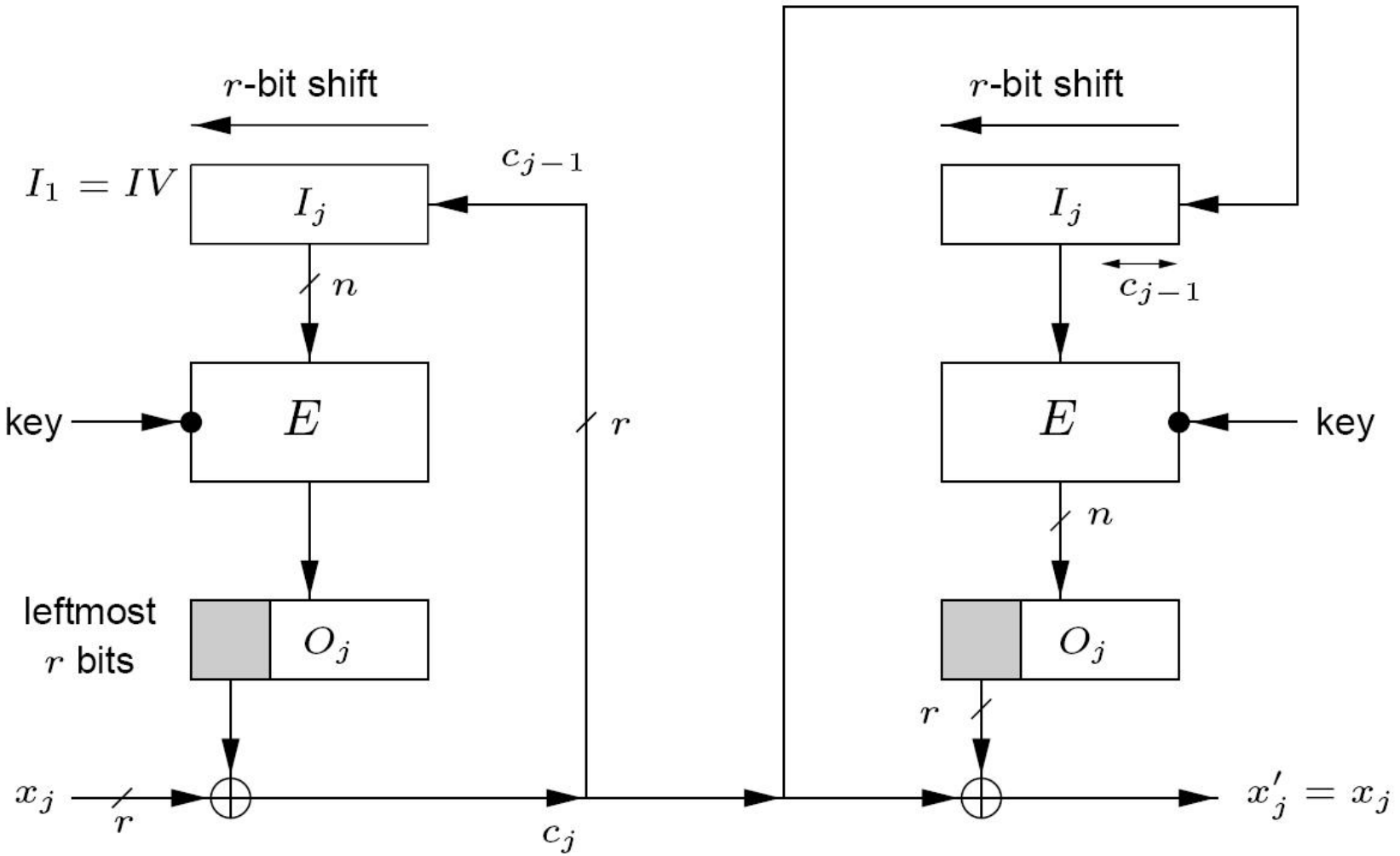
Вопрос

- В OFB, если мы хотим декодировать часть большого сообщения (скажем, начать читать файл не с начала), нам всё равно придётся начать декодировать сначала).
- Как модифицировать OFB, чтобы так не получалось?
- Можно вместо $I_j = E(I_{j-1})$ взять $I_j = E(IV + j)$, например.

CFB

- CFB (cipher feedback): когда нужно иногда отправлять сообщения размером меньше блока, скажем, r. Тогда:
  - берём $I_0 := IV$,
  - считаем код $O_j := E_K(I_j)$,
  - берём t_j как r крайних слева битов O_j,
  - подсчитываем $c_j := t_j \oplus x_j$,
  - сдвигаем $I_{j+1} := 2^r \cdot I_j + c_j \mod 2^n$.

CFB


r-bit shift
$I_1 = IV$
I_j
c_{j-1}
n
key
E
r
leftmost
r bits
O_j
x_j
r
c_j
r-bit shift
I_j
c_{j-1}
E
key
n
O_j
r
$x'_j = x_j$

CFB

- Идентичные сообщения переходят в идентичные коды, только если IV тоже совпадает.
- Блок зависит от предыдущих, переставлять нельзя. Чтобы блок декодировался правильно, надо, чтобы предыдущие $\lceil n/r \rceil$ были правильным.
- Ошибка в блоке c_j вызывает ошибку в $\lceil n/r \rceil$ последующих блоках.
- Каждый запуск E выдаёт только r битов, а не n, как мог бы.

Блочные шифры и MAC'и

- Задача: хотим добавить к сообщению MAC (message authentication code) так, чтобы никто не смог изменить сообщение, не нарушив MAC (целостность).
- Сделать из блочного шифра MAC легко: берём последний блок CBC-режима.
- Более сложный вопрос — как сделать так, чтобы одновременно и закодировать, и целостность обеспечить.

Блочные шифры и MAC'и

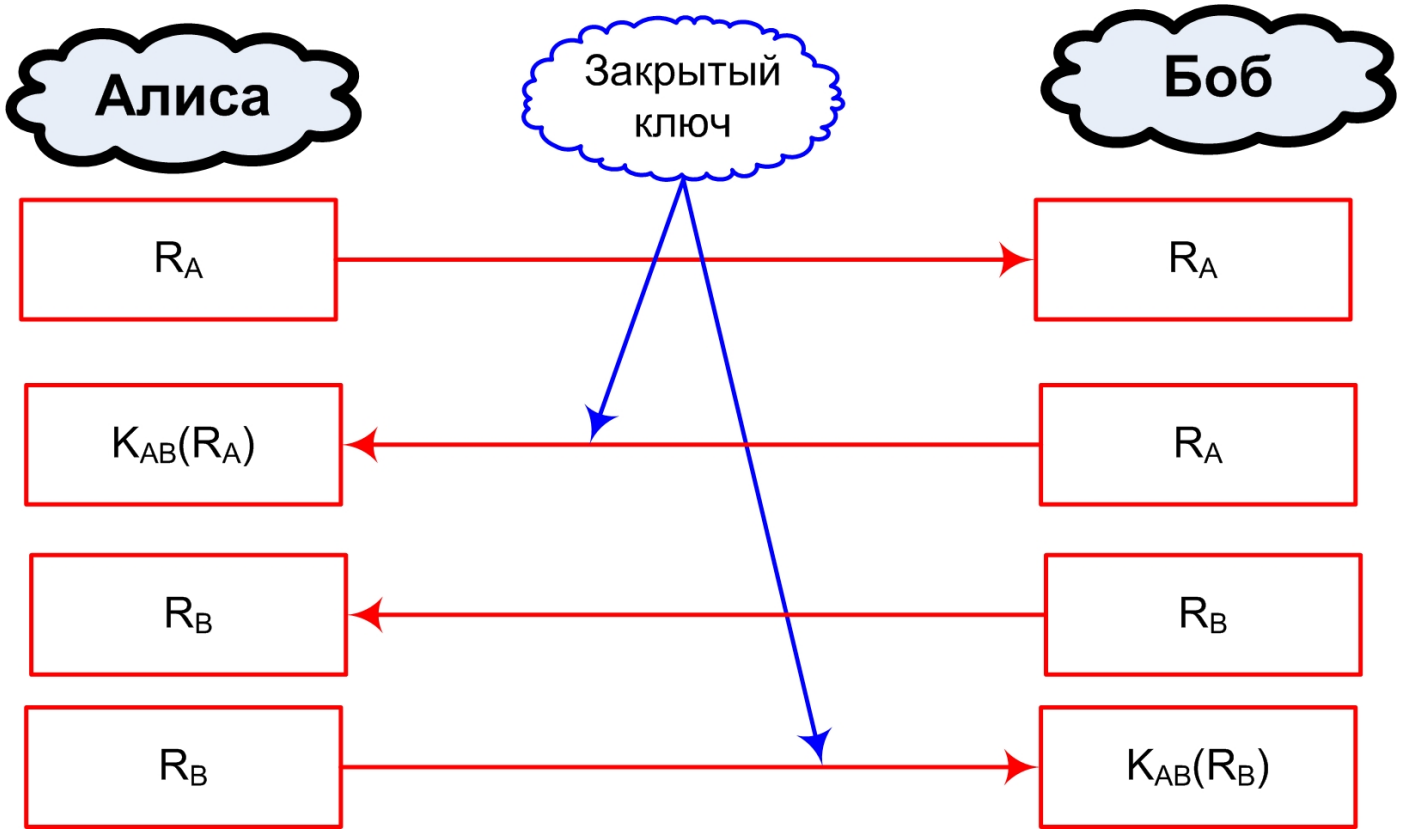
- Если закодировать и взять последний блок, получается бред — просто блок два раза повторили.
- Если взять последний блок, а потом закодировать — тоже бред: последний блок будет XOR'иться с собой, и вместо MAC в конце получится код нуля.
- Можно подсчитать некриптографическую CRC, дописать в конце и закодировать; почти работает, но тоже есть тонкие атаки.
- В общем — считается, что надо получать MAC из другого ключа, чтобы всё было надёжно.

Хеши для криптографии с закрытым ключом

- Оказывается, если у Алисы и Боба есть общий секрет, они могут много чего сделать, не имея алгоритмов для криптографии с закрытым ключом, а только хеш-функции.
- Фактически, можно сделать всё то же самое, что с секретным ключом.

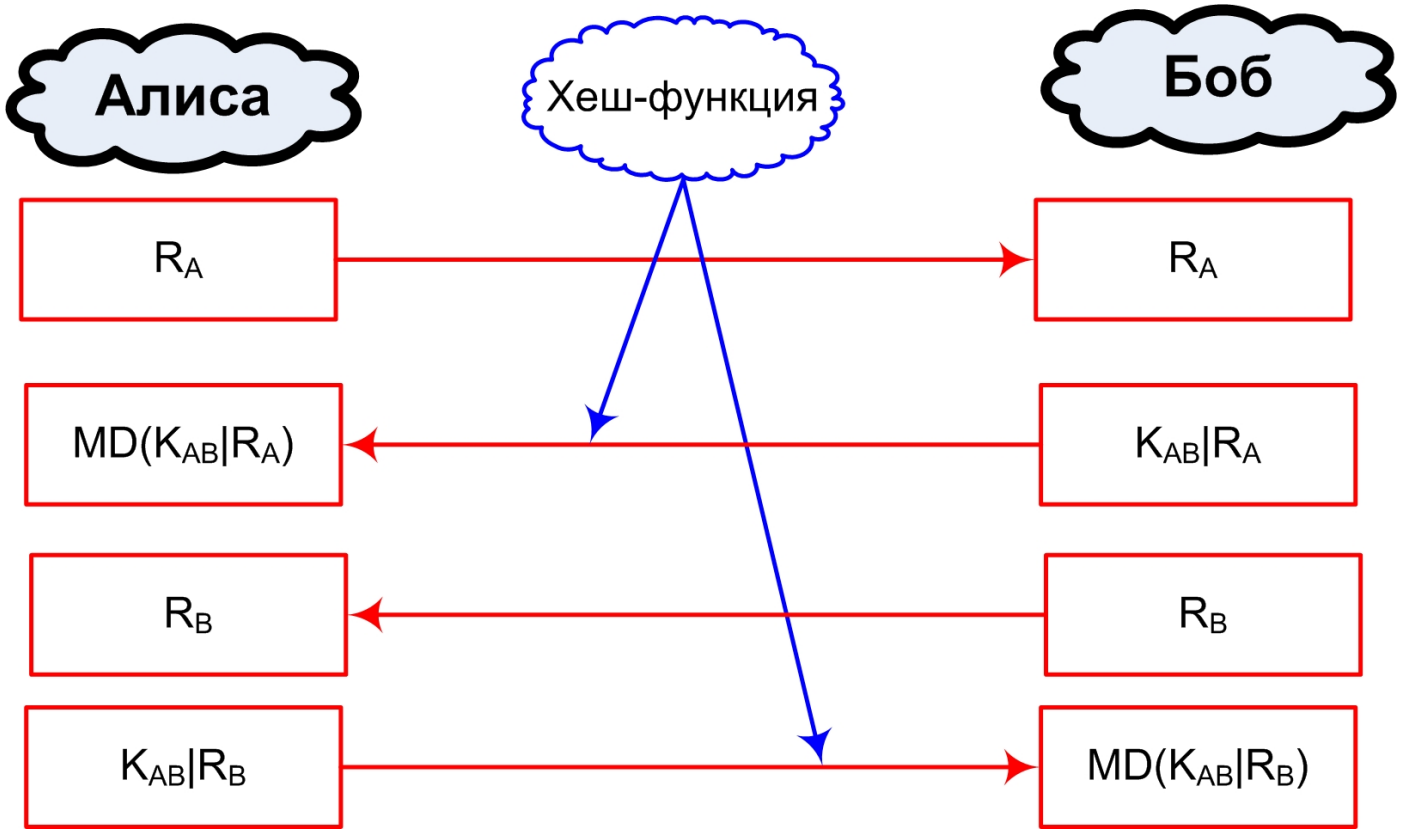
Аутентификация

- Было с секретным ключом:



Аутентификация

- Стало с хеш-функцией:



Создание MAC

- Мы умеем создавать MAC из секретного ключа.
- Можно сделать MAC из хеша — это просто $MD(m)$. Верно?

Создание MAC

- Мы умеем создавать MAC из секретного ключа.
- Можно сделать MAC из хеша — это просто $MD(m)$. Верно?
- Неверно; такой MAC кто угодно сгенерировать может. Поэтому мы делаем то же самое, что и выше, и дописываем
$$\mathrm{MAC} = MD(m \mid K_{AB}).$$
- Тогда Боб может проверить, а никто другой не может.

Кодирование через хеш-функцию

- Кодировать с хеш-функцией любой может. Вот как декодировать - это вопрос... :)
- На самом деле идеи примерно те же, что у схем блочного кодирования.

Кодирование через хеш-функцию

- Можно сгенерировать одноразовый блокнот:

$$b_1 = MD(K_{AB}),\ b_2 = MD(b_1),\ \ldots,\ b_i = MD(b_{i-1}).$$

- Алиса и Боб могут генерировать такую строку заранее, потом применять:

$$c_i = m_i \oplus b_i.$$

Кодирование через хеш-функцию

- Проблема: если угадать/узнать сообщение один раз, можно вычислить код и потом узнать все последующие сообщения.
- Как помочь этой беде?

Кодирование через хеш-функцию

- Проблема: если угадать/узнать сообщение один раз, можно вычислить код и потом узнать все последующие сообщения.
- Как помочь этой беде?
- Например, так:

$$b_1 = MD(K_{AB}),\ b_2 = MD(K_{AB} \mid c_1),\ \ldots,\ b_i = MD(K_{AB} \mid c_{i-1}),$$

$$\text{где } c_i = m_i \oplus b_i.$$

- Как теперь это декодировать?

Кодирование через хеш-функцию

- Проблема: если угадать/узнать сообщение один раз, можно вычислить код и потом узнать все последующие сообщения.
- Как помочь этой беде?
- Например, так:

$$b_1 = MD(K_{AB}),\ b_2 = MD(K_{AB} \mid c_1),\ \ldots,\ b_i = MD(K_{AB} \mid c_{i-1}),$$

$$\text{где } c_i = m_i \oplus b_i.$$

- Как теперь это декодировать?
- Легко: $m_1 = c_1 \oplus MD(K_{AB})$, $m_2 = c_2 \oplus MD(K_{AB} \mid c_1)$, ...

Thank you!

Спасибо за внимание!