Федеральное государственное автономное образовательное учреждение высшего образования
**Национальный исследовательский ядерный университет «МИФИ»**

**Кафедра «Криптология и кибербезопасность»**

# Криптографические протоколы

## курс лекций

***Запечников Сергей Владимирович,***
*профессор кафедры «Криптология*
*и кибербезопасность» НИЯУ МИФИ*

***Москва – 2018***

**Криптографические протоколы**

***курс лекций***

# Лекция 3.
# Обеспечение безопасности доступа к базам данных и облачным хранилищам данных

***19 февраля 2018 г.***

# Задачи обеспечения доступа к информационным массивам, хранимым дистанционно

1. **Обеспечение высокой доступности → безопасное размещение информационного массива в распределенной среде.**

2. **Обеспечение целостности → дистанционный контроль целостности файлов, баз данных и массивов, размещенных в облачной среде.**

3. **Обеспечение конфиденциальности → конфиденциальный поиск в базе данных.**

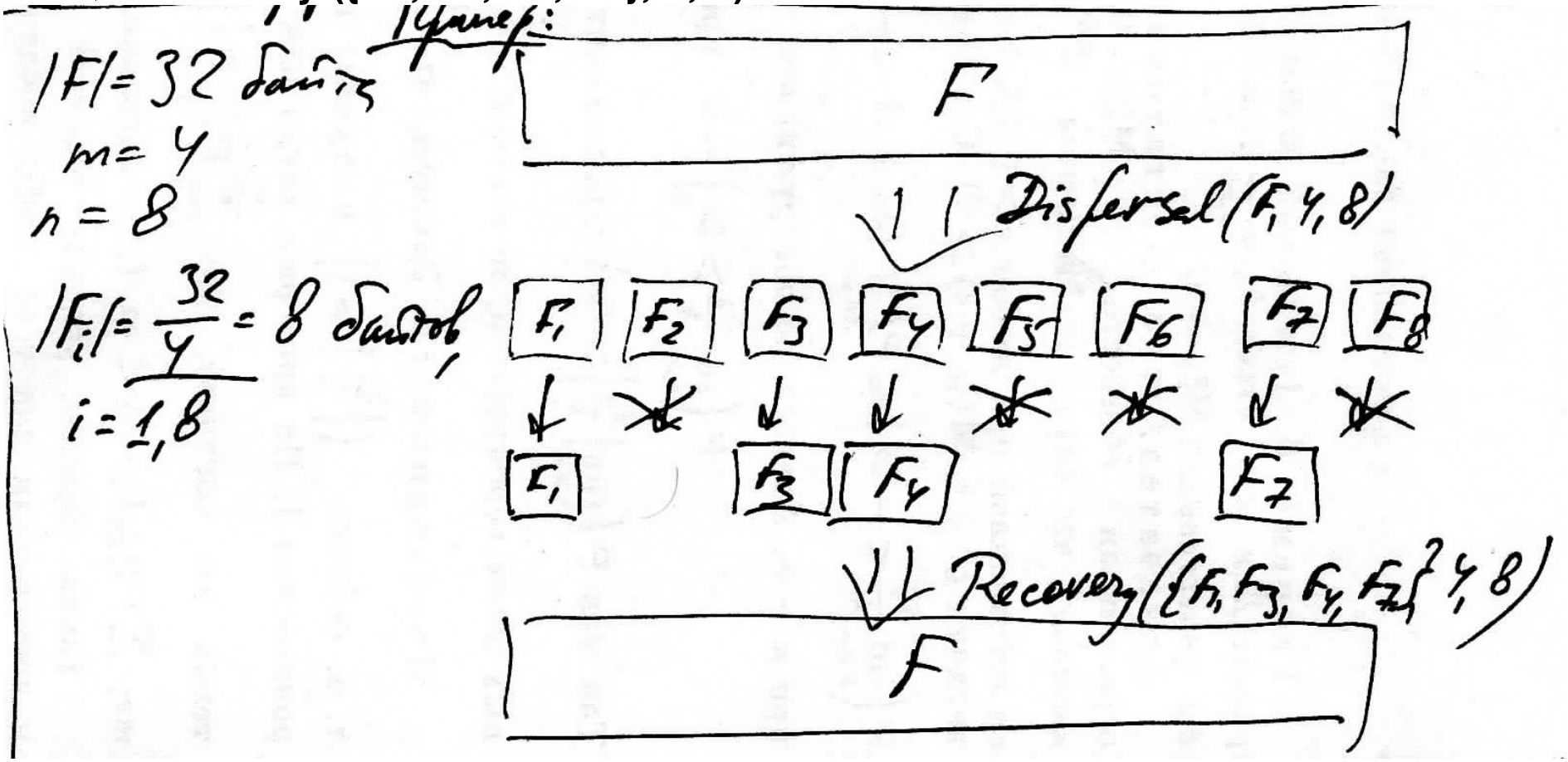
# Безопасное размещение информационного массива в распределенной среде

**Рассмотрим следующую задачу. Пусть *F* – некоторый массив данных, длина этого массива равна *N* байтов: $|F| = N$, *n* – количество устройств хранения данных (УХД), на которых этот массив может быть размещен. Необходимо предложить такой метод распределения блоков массива по *n* УХД, который обеспечивал бы возможность полного восстановления всего массива по информации, содержащейся на любых *m* УХД, где $1 \le m \le n$.**

***Метод решения* этой задачи заключается в следующем. Массив *F* разбивается на *n* фрагментов: $F_i$, $i = \overline{1,n}$, где $|F_i| = \frac{|F|}{m}$ (таким образом, $m|N$) при помощи некоторого алгоритма Dispersal (F, m, n), после чего полученные таким образом фрагменты массива распределяются по УХД. В любой момент времени после распределения фрагментов массива по УХД исходный массив *F* может быть восстановлен при помощи алгоритма Recovery ($\{F_{i_j}, j = \overline{1,m}, 1 \le i_j \le n\}$, m, n). Заметим, что пока мы рассматриваем эту задачу, не выдвигая требований к обеспечению конфиденциальности массива, т.е. предполагается, что он не содержит информации ограниченного распространения.**

# Пример распределения массива

**Пусть $|F| = 32$ байта, $m$=4, $n$=8. Тогда $|F_i| = 32/4 = 8$ байтов, $i = \overline{1,8}$. Массив распределяется при помощи протокола Dispersal (F, 4, 8), собирается (в данном случае) – при помощи Recovery ({F1, F3, F4, F7}, 4, 8).**



# Схема IDA (1)

**Для решения этой задачи M.O. Rabin в 1989 г. предложил алгоритм, который получил название *IDA (Information Dispersal Algorithm)*.**

***Алгоритм распределения блоков массива*** **Dispersal (F, m, n):**

**1. Массив F представляется в виде последовательности байтов, где каждый байт интерпретируется как элемент поля характеристики 2:** $F = b_1, b_2, ..., b_N$ , $b_i \in \mathrm{GF}(2^8)$ **. Существенными моментами для дальнейшего описания метода здесь являются свойства элементов поля: во-первых, любой ненулевой элемент поля имеет обратный, во-вторых, поле замкнуто относительно операций сложения и умножения.**

**2. Массив представляется в матричной форме в виде последовательности столбцов:** $F = (b_1, ..., b_m), (b_{m+1}, ..., b_{2m}), ..., (b_{N-m+1}, ..., b_N)$ **. Вводятся обозначения:** $\overrightarrow{S_i} = (b_{(i-1)m+1}, ..., b_{im})$, $i = \overline{1, N/m}$ **. Получается матрица** $M = \left[\overrightarrow{S_1}\overrightarrow{S_2}...\overrightarrow{S_{N/m}}\right]$ **размерности** $m \times \frac{N}{m}$ **.**

# Схема IDA (2)

**3. Выбирается матрица $A$ размерности $n \times m$, состоящая из таких строк $\vec{a}_i = (a_{i1},...,a_{im})$, $i = \overline{1,n}$, что любые *m* различных векторов линейно независимы. На эту роль подходит матрица Вандермонда, где $m \le n$, и все $x_i$ – попарно различные ненулевые элементы поля $\mathrm{GF}(2^8)$:**

$$A = \begin{bmatrix} \vec{a}_1 \\ \vec{a}_2 \\ \dots \\ \vec{a}_n \end{bmatrix}, \qquad A = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{m-1} \\ 1 & x_3 & x_3^2 & \dots & x_3^{m-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{m-1} \\ 1 & x_n & x_n^2 & \dots & x_n^{m-1} \end{bmatrix}.$$

**В ней любые *m* различных строк линейно независимы, так что любая матрица, составленная из *m* различных строк матрицы Вандермонда, обратима. С подобной матрицей мы уже встречались при рассмотрении пороговых СРС.**

# Схема IDA (3)

**4. Фрагменты, размещаемые на разных УХД, вычисляются следующим образом:**

$$A \cdot M = \begin{bmatrix} \overrightarrow{a_1} \\ \overrightarrow{a_2} \\ \dots \\ \overrightarrow{a_n} \end{bmatrix} \cdot \begin{bmatrix} \overrightarrow{S_1} & \overrightarrow{S_2} & \dots & \overrightarrow{S_{\frac{N}{m}}} \end{bmatrix} = \begin{bmatrix} \overrightarrow{a_1}\overrightarrow{S_1} & \overrightarrow{a_1}\overrightarrow{S_2} & \dots & \overrightarrow{a_1}\overrightarrow{S_{\frac{N}{m}}} \\ \overrightarrow{a_2}\overrightarrow{S_1} & \overrightarrow{a_2}\overrightarrow{S_2} & \dots & \overrightarrow{a_2}\overrightarrow{S_{\frac{N}{m}}} \\ \dots & \dots & \dots & \dots \\ \overrightarrow{a_n}\overrightarrow{S_1} & \overrightarrow{a_n}\overrightarrow{S_2} & \dots & \overrightarrow{a_n}\overrightarrow{S_{\frac{N}{m}}} \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ \dots \\ F_n \end{bmatrix}.$$

**Каждый элемент произведения матриц равен:** $\overrightarrow{a_i} \cdot \overrightarrow{S_k} = a_{i1} b_{(k-1)m+1} + \dots + a_{im} b_{km} \in \mathrm{GF}(2^8)$ .

**Следовательно,** $F_i \in \left[\mathrm{GF}(2^8)\right]^{\frac{N}{m}}$ .

# Схема IDA (4)

***Алгоритм сборки массива из части фрагментов*** **Recovery (**$\{F_{i_j}, j=\overline{1,m}, 1 \le i_j \le n\}$**, m, n):**

**Пусть даны *m* фрагментов массива, размещенные на *m* различных УХД:** $\{F_{i_j}, j=\overline{1,m}, 1 \le i_j \le n\}$**. Тогда поскольку**

$$\begin{bmatrix} F_{i1} \\ F_{i2} \\ \dots \\ F_{im} \end{bmatrix} = \begin{bmatrix} \overrightarrow{a_{i1}} \\ \overrightarrow{a_{i2}} \\ \dots \\ \overrightarrow{a_{im}} \end{bmatrix} \cdot M = A' \cdot M ,$$

**и матрица *A* обратима, матрица *M* может быть восстановлена из этих фрагментов:**

$$\begin{bmatrix} \overrightarrow{a_{i1}} \\ \overrightarrow{a_{i2}} \\ \dots \\ \overrightarrow{a_{im}} \end{bmatrix}^{-1} \times \begin{bmatrix} F_{i1} \\ F_{i2} \\ \dots \\ F_{im} \end{bmatrix} = M .$$

# Пример применения схемы IDA (1)

**Пусть, как и в предыдущем примере, $|F| = N = 32$ байта, *m*=4, *n*=8. Распределение массива по алгоритму IDA происходит следующим образом:**

$$F = b_1, b_2, ..., b_{32} = (b_1, ..., b_4), (b_5, ..., b_8), ..., (b_{29}, ..., b_{32}),$$

$$M = \begin{bmatrix} \overrightarrow{S_1} & \overrightarrow{S_2} & \dots & \overrightarrow{S_8} \end{bmatrix} = \begin{bmatrix} b_1 & b_5 & \dots & b_{29} \\ b_2 & b_6 & \dots & b_{30} \\ b_3 & b_7 & \dots & b_{31} \\ b_4 & b_8 & \dots & b_{32} \end{bmatrix},$$

$$A = \begin{bmatrix} \overrightarrow{a_1} \\ \overrightarrow{a_2} \\ \dots \\ \overrightarrow{a_8} \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \dots & \dots & \dots & \dots \\ 1 & x_8 & x_8^2 & x_8^3 \end{bmatrix}.$$

**как видим, матрица *M* имеет размерность $4 \times 8$, матрица *A* – $8 \times 4$. Следовательно, эти матрицы можно перемножить:**

$$A \cdot M = \begin{bmatrix} \overrightarrow{a_1} \\ \overrightarrow{a_2} \\ \dots \\ \overrightarrow{a_8} \end{bmatrix} \cdot \begin{bmatrix} \overrightarrow{S_1} & \overrightarrow{S_2} & \dots & \overrightarrow{S_8} \end{bmatrix} =$$

# Пример применения схемы IDA (2)

$$= \begin{bmatrix} \overrightarrow{a_1}\overrightarrow{S_1} = b_1 + x_1 b_2 + x_1^2 b_3 + x_1^3 b_4 & \ldots & \overrightarrow{a_1}\overrightarrow{S_8} = b_{29} + x_1 b_{30} + x_1^2 b_{31} + x_1^3 b_{32} \\ \ldots\ldots\ldots & \ldots & \ldots\ldots\ldots \\ \overrightarrow{a_8}\overrightarrow{S_1} = b_1 + x_8 b_2 + x_8^2 b_3 + x_8^3 b_4 & \ldots & \overrightarrow{a_8}\overrightarrow{S_8} = b_{29} + x_8 b_{30} + x_8^2 b_{31} + x_8^3 b_{32} \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ \ldots \\ F_8 \end{bmatrix}.$$

**Таким образом,** $|F_i| = 8$ **байтов, а** $\sum_{i=1}^{8} |F_i| = \frac{n}{m} \cdot N = 64$ **байта.**

**Восстановление массива по четырем выбранным фрагментам** $\{F_1, F_3, F_4, F_7\}$ **осуществляется следующим образом:**

$$\begin{bmatrix} F_1 \\ F_3 \\ F_4 \\ F_7 \end{bmatrix} = \begin{bmatrix} \overrightarrow{a_1}\overrightarrow{S_1} & \overrightarrow{a_1}\overrightarrow{S_2} & \ldots & \overrightarrow{a_1}\overrightarrow{S_8} \\ \overrightarrow{a_3}\overrightarrow{S_1} & \overrightarrow{a_3}\overrightarrow{S_2} & \ldots & \overrightarrow{a_3}\overrightarrow{S_8} \\ \overrightarrow{a_4}\overrightarrow{S_1} & \overrightarrow{a_4}\overrightarrow{S_2} & \ldots & \overrightarrow{a_4}\overrightarrow{S_8} \\ \overrightarrow{a_7}\overrightarrow{S_1} & \overrightarrow{a_7}\overrightarrow{S_2} & \ldots & \overrightarrow{a_7}\overrightarrow{S_8} \end{bmatrix} = \begin{bmatrix} \overrightarrow{a_1} \\ \overrightarrow{a_3} \\ \overrightarrow{a_4} \\ \overrightarrow{a_7} \end{bmatrix} \cdot M,$$

**т.е.**

$$M = \begin{bmatrix} \overrightarrow{a_1} \\ \overrightarrow{a_3} \\ \overrightarrow{a_4} \\ \overrightarrow{a_7} \end{bmatrix}^{-1} \times \begin{bmatrix} F_1 \\ F_3 \\ F_4 \\ F_7 \end{bmatrix}.$$

# Схема, обеспечивающая доступность и целостность информационного массива при утрате части устройств хранения данных (1)

***Схема, обеспечивающая доступность и целостность информационного массива при утрате не более t из n УХД, где*** $t < n$. **Пусть *U* – владелец массива, *GW* – шлюз для доступа в облачную среду (сервер облачного сервиса),** $V_i, V_j$ **– серверы или УХД в облачной среде. Схема включает в себя два протокола и один алгоритм:**

- **Deposit – протокол, в котором пользователь *U* контактирует со шлюзом GW, размещает файл *F* и получает подтверждение этого;**
- **Dispersal – алгоритм, выполняя который УХД перераспределяют между собой фрагменты помещенного на хранение файла (алгоритм выполняется каждым УХД автономно, но синхронно с остальными УХД);**
- **Retrieval – протокол, в котором пользователь *U* контактирует со шлюзом GW и забирает обратно файл *F*.**

# Схема, обеспечивающая доступность и целостность информационного массива при утрате части устройств хранения данных (2)

**Для построения этой криптосхемы необходимо выбрать любую схему ЦП $(Gen, Sign, Ver)$, для которой генерируются три пары ключей: $(sk_U, pk_U)$ – ключи владельца файла, $(sk_{V_i}, pk_{V_i})$, $i = \overline{1,n}$ – ключи УХД и $(sk_{GW}, pk_{GW})$ – ключи шлюза. Сертификаты всех открытых ключей создаются УЦ. В качестве альтернативы могут быть выбраны две любые схемы ЦП, одна из которых обычная, другая – пороговая. Ключи обычной схемы ЦП генерируются для пользователя: $(sk_U, pk_U)$, ключи пороговой схемы ЦП – для всех УХД: $\left(sk_V \xleftrightarrow{(t,n)} \left(sk_{V_1}, sk_{V_2}, \ldots, sk_{V_n}\right), pk_V\right)$, причем каждому из них выдается одна доля секретного ключа и открытый ключ. В этом случае шлюзу ключи вообще не нужны. Кроме того, выбираются $H(\cdot)$ – односторонняя хэш-функция с трудно обнаруживаемыми коллизиями и $A$ – матрица Вандермонда для схемы IDA. Хэш-функция и матрица считаются общеизвестными. Наконец, фиксируется $t = n - m$ – предельно допустимый порог утраченных УХД, который задается заранее.**

# Схема, обеспечивающая доступность и целостность информационного массива при утрате части устройств хранения данных (3)

**Протокол Deposit:**

**(1)** $U \rightarrow GW$ **:** $F, Sign_{sk_U}(F)$

**(2)** $GW \rightarrow V_j, \forall j$ **:** $F, Sign_{sk_U}(F)$

**(3)** $V_j, \forall j \rightarrow V_i, \forall i$**:** $F, Sign_{sk_U}(F)$

**(4)** $v_j$ **из шага (2)** $\rightarrow GW$**:** $Sign_{sk_{V_j}}(U, F)$

**(5)** $GW \rightarrow U$ **:** $Sign_{sk_{GW}}(U, F)$**.**

**Для протокола с использованием пороговой подписи шаг (4) будет выглядеть следующим образом:**

**(4)** $V_j$ **из шага (2)** $\rightarrow GW$**:** $Sign_{sk_{V_j}}(U, F)$**, после чего** $GW$ **собирает из фрагментов целую подпись:** $Sign_{sk_V}(U, F)$**.**

# Схема, обеспечивающая доступность и целостность информационного массива при утрате части устройств хранения данных (4)

**Алгоритм Dispersal:**

**Каждый** $V_i$, $i = \overline{1,n}$ **выполняет следующие действия:**

**а) для** $\forall j$, $j = \overline{1,n}$ **вычисляет**

$$A \cdot M = \begin{bmatrix} F_1 \\ F_2 \\ \dots \\ F_n \end{bmatrix},$$

**где** $A$ **– матрица IDA,** $M$ **– массив** $F$ **в матричной форме;**

**б) для** $\forall j$, $j = \overline{1,n}$ **вычисляет** $H\left(F_j\right)$**;**

**в) сохраняет на своем УХД совокупность данных**

$$\left\{F_i; \left\{H\left(F_j\right), j = \overline{1,n}\right\}; Sign_{sk_U}\left(F\right)\right\}.$$

# Схема, обеспечивающая доступность и целостность информационного массива ... (5)

**Протокол Retrieval:**

**(1)** ***U*** **→** ***GW*****:** $Sign_{sk_U}\left(F\right)$

**(2)** ***GW*** **→** $V_j, \forall j$ **:** $Sign_{sk_U}\left(F\right)$

**(3)** $V_j, \forall j$ **→** ***GW*****:** $F_j, \left\{H\left(F_i\right), i=\overline{1,n}\right\}$

**(4)** ***GW*** **выполняет следующие действия:**

**а)** для $\forall j$: **устанавливает в качестве** $H\left(F_j\right)$ **то значение, которое принимает большинство из полученных** $H\left(F_j\right)$**;**

**б) формирует** ***G*** **– множество индексов тех УХД, от которых получены верные хэш-коды: первоначально множество** ***G*** **устанавливается в** $\varnothing$ **(пустое множество), затем если присланный** ***j*****-м УХД** $H\left(F_j\right)$ **совпал с установленным значением, то** $G \leftarrow G \cup \left\{j\right\}$**;**

**в) вычисляет** $M = A^{-1} \cdot \begin{bmatrix} F_{i1} \\ F_{i2} \\ \dots \\ F_{im} \end{bmatrix}$,

**где** ***M*** **– файл** ***F*****, записанный в матричной форме;**

**(5)** ***GW*** **→** ***U*****:** ***F*****.**

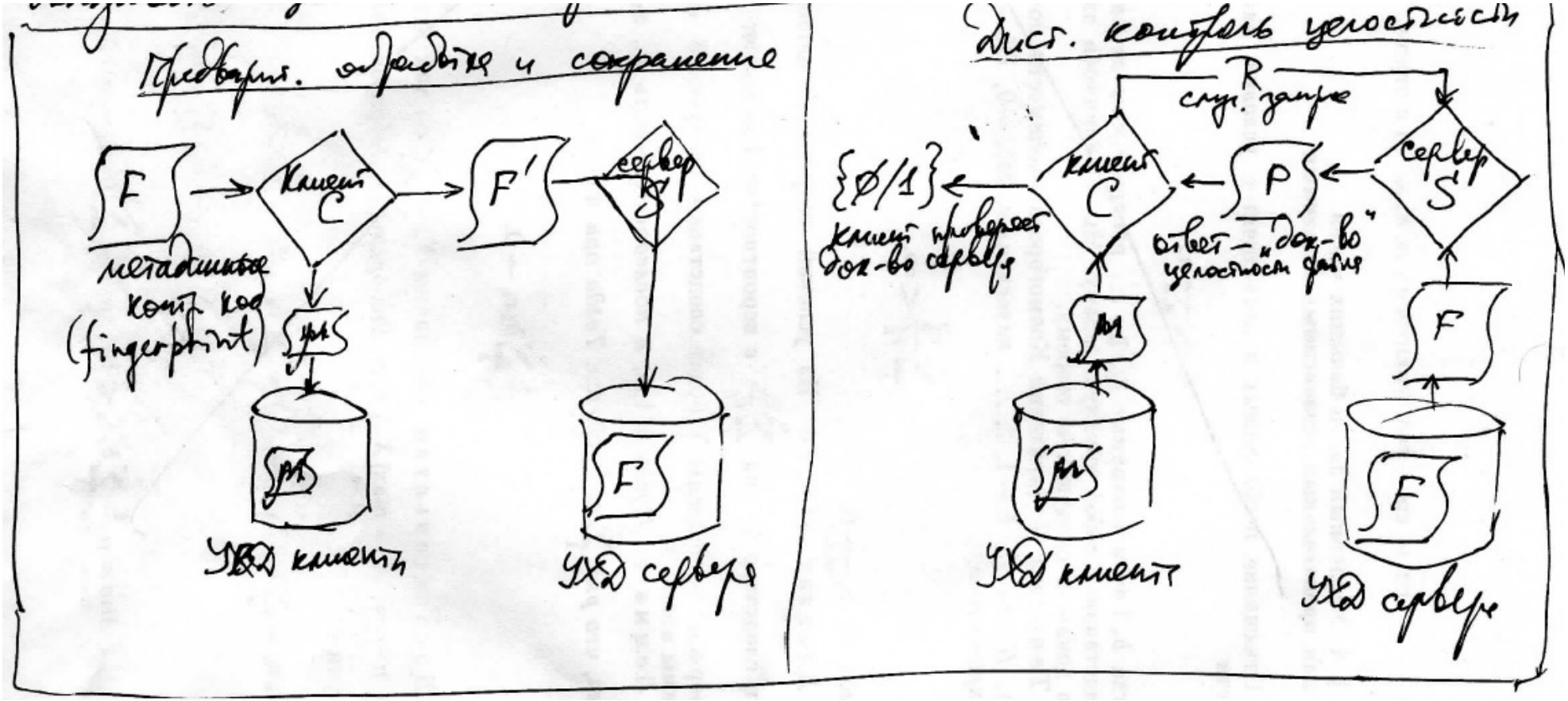
# Дистанционный контроль целостности информационных массивов (1)

**Пусть информационный массив (файл) *F*, состоящий из некоторого количества *n* блоков, принадлежит клиенту *C*, но хранится не у него, а у внешнего провайдера. Необходимо обеспечить для клиента возможность периодического контроля сохранности этого файла. Иными словами, клиент должен иметь возможность проверить, что блоки файла действительно хранятся на УХД провайдера, и сохраняется их целостность, не загружая весь файл к себе.**

***Идея решения задачи.*** **Клиент *C* предварительно обрабатывает хранящийся у него локально файл, генерирует метаданные, в том числе контрольный код, который также сохраняет локально. После этого он пересылает файл *F* на сервер *S*, получает подтверждение сохранения и удаляет его локальную копию. Далее сервер хранит файл на своем УХД и отвечает на запросы клиента. Если файл содержит информацию ограниченного распространения, он должен быть зашифрован.**

**Запросы клиента могут требовать передать ему часть файла для получения каких-либо сведений или просто предназначаться для контроля целостности. Такой контроль может быть осуществлен и сразу после отсылки файла на сервер, перед удалением локальной копии. Сервер может пытаться обмануть клиента, утверждая, что у него хранится файл, который на самом деле удален или поврежден. В целях экономии затрат на выполнение протокола допускается вероятностный контроль, т.е. результат контроля может доказывать целостность файла с вероятностью $P < 1$.**

# Дистанционный контроль целостности информационных массивов (2)



# Схема дистанционного контроля целостности массива (1)

**Пусть *C* – клиент, *S* – сервер, $F = (m_1, ..., m_n)$ – информационный массив (файл), состоящий из упорядоченной последовательности блоков. Вводится понятие гомоморфного проверяемого тэга (homomorphic verifiable tag – HVT). Это пара чисел $(T_{W,m}, W)$, где *W* – случайная величина, которая не повторяется для различных блоков. Свойство гомоморфности заключается в том, что существует правило, пользуясь которым по заданным $(T_{W_i,m_i}, W_i)$ и $(T_{W_j,m_j}, W_j)$ можно найти $T_{\{W_i,W_j\},m_i+m_j}$ – тэг для сообщения $m_i + m_j$.**

# Схема дистанционного контроля целостности массива (2)

***Схема дистанционного контроля целостности*** **состоит из четырех алгоритмов и двух протоколов:**

- **KeyGen** $\rightarrow (pk, sk)$ **– алгоритма генерации ключей, выполняемого клиентом;**
- **TagBlock** $(sk, m) \rightarrow (T_{W,m}, W)$ **– алгоритма генерации проверочных метаданных, т.е. гомоморфного проверяемого тэга, выполняемого клиентом;**
- **GenProof** $(pk, F, \text{chal}, \Sigma) \rightarrow V$ **– выполняемого сервером алгоритма генерации доказательства целостности файлов, где chal – запрос клиента, а** $\Sigma$ **– набор проверочных метаданных для файла F, сгенерированный сервером;**
- **CheckProof** $(pk, sk, \text{chal}, V) \rightarrow \{\text{"success"}, \text{"failure"}\}$ **– выполняемого клиентом алгоритма проверки доказательства, вырабатываемого сервером, который позволяет сделать вывод о сохранности либо нарушении целостности файла;**
- **Setup – протокола загрузки файла клиентов на сервер;**
- **Challenge – протокола проверки целостности файла** ***F*****.**

# Реализация схемы дистанционного контроля целостности массива (1)

**Рассмотрим реализацию схемы.**

**Пусть** $p = 2p' + 1$, $q = 2q' + 1$ **– простые числа,** $N = p \cdot q$, ***g*** **– образующий элемент циклической подгруппы** $\mathbf{Z}_N^*$ **порядка** $p'q'$ **(т.е.** $QR_N$ **– это множество квадратичных вычетов по модулю *N*). Образующий элемент *g* можно получить так:** $g = a^2$, **где** $a \overset{R}{\leftarrow} \mathbf{Z}_N^*$, **таким образом, что** $\text{НОД}(a \pm 1, N) = 1$. **Экспоненцирование выполняется по модулю *N*. Пусть** $h : \{0,1\}^* \to QR_N$ **– однонаправленная хэш-функция с трудно обнаруживаемыми коллизиями,** $f, w$ **– псевдослучайные функции (псевдослучайные генераторы),** $\pi$ **– псевдослучайная перестановка (блочный шифр),** $H(\cdot)$ **– криптографическая хэш-функция.**

**KeyGen:**

**Генерируется пара ключей:** $pk = (N, e, g)$, $sk = (N, d, v)$, **где** $(N, e)$ **и *d* – соответственно открытый и секретный ключи схемы RSA, а** $v \overset{R}{\leftarrow} \{0,1\}^{\kappa}$.

# Реализация схемы дистанционного контроля целостности массива (2)

**TagBlock**$(sk, m, i)$**:**

**1. Вычисляется** $W_i = w_v(i)$**.**

**2. Вычисляется** $T_{W_i,m} = \left(h(W_i) \cdot g^m\right)^d \bmod N$**.**

**3. В качестве проверочных метаданных принимается** $\left(T_{W_i,m_i}, W_i\right)$**.**

**GenProof**$\left(pk, F = (m_1,...,m_n), \text{chal}, \Sigma = \left\{T_{W_1,m_1},....,T_{W_n,m_n}\right\}\right)$**:**

**1.Пусть** $(N, e, g) = pk$ **и** $(c, k_1, k_2, s) = \text{chal}$**. Тогда для всех** $j = \overline{1,c}$**:**

**а) вычисляются индексы блоков, для которых будет выполняться проверка:** $i_j = \pi_{k_1}(j)$**;**

**б) вычисляются коэффициенты** $a_j = f_{k_2}(j)$**.**

**2. Вычисляется** $T = T_{W_{i_1},m_{i_1}}^{a_1} \cdots T_{W_{i_c},m_c}^{a_c} = \left(h\left(W_{i_1}\right)^{a_1} \cdots h\left(W_{i_c}\right)^{a_c} \cdot g^{a_1 m_{i_1} + ... + a_c m_{i_c}}\right)^d \bmod N$**.**

**3. Вычисляется** $\rho = H\left(\left(g^s\right)^{a_1 m_{i_1} + ... + a_c m_{i_c}} \bmod N\right)$**.**

**4. В качестве доказательства целостности принимается** $V = (T, \rho)$**.**

# Реализация схемы дистанционного контроля целостности массива (3)

**CheckProof** $(pk, sk, \text{chal}, V)$**:**

**1. Пусть** $(N, e, g) = pk$**,** $(N, d, v) = sk$**,** $(c, k_1, k_2, s) = \text{chal}$**,** $(T, \rho) = V$**.**

**2. Пусть** $\tau = T^e$**. Для всех** $j = \overline{1, c}$ **в цикле вычисляется:**

$$\left\{ i_j = \pi_{k_1}(j), W_{i_j} = w_v(i_j), a_j = f_{k_2}(j), \tau := \frac{\tau}{h\left(W_{i_j}\right)^{a_j}} \bmod N \right\}.$$

**(как результат получится** $\tau = g^{a_1 m_{i_1} + \ldots + a_c m_{i_c}} \bmod N$**).**

**3.Если** $H(\tau^s) = \rho \bmod N$**, то выдается сообщение "success", в противном случае – "failure".**

# Реализация схемы дистанционного контроля целостности массива (4)

**Setup:**

**1. Клиент *C*, обладающий файлом *F*, выполняет алгоритм KeyGen** $\rightarrow (pk, sk)$**.**

**2. Клиент вычисляет проверочные метаданные: TagBlock** $(sk, m_i) \rightarrow \left(T_{W_i, m_i}, W_i\right)$ **для** $\forall i = \overline{1, n}$**.**

**3. Клиент сохраняет пару ключей** $(sk, pk)$ **на своем УХД.**

**4. Клиент *C* отсылает серверу *S* сообщение** $\left\{pk, F, \Sigma = \left\{T_{W_1, m_1}, \ldots, T_{W_n, m_n}\right\}\right\}$**.**

**5. Клиент удаляет *F* и** $\Sigma$ **со своего локального диска.**

# Реализация схемы дистанционного контроля целостности массива (5)

**Challenge:**

**Клиент *C* требует доказательства целостности с выделенных блоков файла *F*, где** $1 \le c \le n$**. Для этого:**

**1. Клиент *C* генерирует запрос chal=**$(c, k_1, k_2, s)$**, где** $k_1 \xleftarrow{R} \{0,1\}^{\kappa}$**,** $k_2 \xleftarrow{R} \{0,1\}^{\kappa}$**,** $s \xleftarrow{R} \mathbf{Z}_N^*$**, и посылает этот запрос серверу *S*.**

**2. Сервер *S* выполняет алгоритм GenProof**$\left(pk, F = (m_1, ..., m_n), \text{chal}, \Sigma = \left\{T_{W_1, m_1}, ..., T_{W_n, m_n}\right\}\right) \to V$ **и посылает доказательство *V* клиенту *C*.**

**3. Клиент *C*, используя chal, проверяет доказательство *V* путем выполнения алгоритма CheckProof**$(pk, sk, \text{chal}, V)$**.**

# Свойства вероятностного контроля целостности (1)

***Утверждение.*** **Предположим, что *S* удалил *t* из *n* блоков файла *F*. Пусть *C* – количество различных блоков, для которых *C* просит представить доказательство целостности. Обозначим через *X* случайную переменную, которая определяет количество блоков, выбранных *C*, которые совпадают с блоками, удаленными *S*. Тогда вероятность $P_X$ того, что хотя бы один блок, выбранный *C*, совпадет с одним из блоков, удаленных *S*, оценивается неравенством:**

$$1-\left(\frac{n-t}{n}\right)^c \le P_X \le 1-\left(\frac{n-c+1-t}{n-c+1}\right)^c.$$

# Свойства вероятностного контроля целостности (2)

***Доказательство:***

$$P_X = P(X \geq 1) = 1 - P(X = 0) = 1 - \frac{n-t}{n} \cdot \frac{n-1-t}{n-1} \cdot \frac{n-2-t}{n-2} \cdots \frac{n-c+1-t}{n-c+1}.$$

**Так как** $\frac{n-i-t}{n-i} \geq \frac{n-i-1-t}{n-i-1}$, $0 \leq i < c$, **то отсюда следует, что**

$$1 - \left(\frac{n-t}{n}\right)^c \leq P_X \leq 1 - \left(\frac{n-c+1-t}{n-c+1}\right)^c. \square$$

$P_X$ **показывает вероятность того, что если *S* удалил *t* блоков файла, то *C* сможет это обнаружить, послав запрос на проверку *c* блоков файла.**

**Особенность метода заключается в том, что когда *S* удалит некоторую долю блоков файла, то *C* сможет обнаружить этот факт с определенной вероятностью, запрашивая постоянное число блоков, независимо от общего числа блоков: так, если *t*=1%, то *C* запрашивает 460 блоков и 300 блоков для того, чтобы достичь вероятности $P_X$, равной 99% и 95% соответственно.**

# Задачи конфиденциального доступа к базе данных

**1. *Схемы поиска по шифрованным и сжатым данным* – это такие схемы, которые предполагают поиск в БД по ключевым словам или иным поисковым ключам. К этим схемам относятся:**

- **безопасные индексы (пример – фильтр Блюма);**
- **симметричные и асимметричные схемы шифрования с возможностью поиска.**

**2. *Схемы конфиденциального выбора информации из БД* (PIR – Private Information Retrieval) – это схемы, главной задачей которых является обеспечение невозможности для стороннего наблюдателя, включая самого провайдера облачного сервиса, идентифицировать те записи БД, из которых происходит выборка информации при обращении к БД.**

**4. *Схемы конфиденциального произвольного доступа к массиву* (ORAM – Oblivious Random Access Memory) – это схемы, обладающие тем и же свойствами, что и PIR, но обеспечивающие эти свойства как для операций чтения, так и для операций записи в память.**

# Фильтры Блума (1)

**Фильтры Блума (*B.H. Bloom, 1970*) – это метод построения структур данных для проверки вхождения некоторого элемента во множество, достаточно эффективный в смысле затрат процессорного времени и памяти.**

**Пусть $S = \{s_1, ..., s_n\}$ – множество из *n* элементов, *m* – размер вспомогательного массива, предназначенного для описания этого множества, $m \ll n$ . Все элементы массива изначально установлены в 0. Фильтр использует *r* независимых хэш-функций $f_1, ..., f_r$, где $f_i : \{0,1\}^* \to \{1, ..., m\}$ для $1 \le i \le r$. Для каждого элемента $s \in S$ биты массива на позициях $f_1(s), ..., f_r(s)$ устанавливаются в 1. Позиция может быть установлена в 1 несколько раз, но отмечается это только в первый раз. Для того, чтобы проверить принадлежность элемента *a* к множеству *S*, проверяются биты на позициях $f_1(a), ..., f_r(a)$. Если все проверенные биты равны 1, то предполагается, что *a* входит во множество. Однако существует возможность ложного вхождения, когда на основании проверки битов делается вывод, что *a* входит в *S*, а на самом деле его там нет. Ложные вхождения могут случаться потому, что каждая позиция может быть установлена в 1 элементом, отличным от *a*. С другой стороны, если какой-либо из проверенных битов равен 0, то тогда *a* точно не входит в *S*. Фильтр Блума можно использовать для каждого документа в качестве индекса для отслеживания присутствующих в нем слов, тем самым реализуя полнотекстовый поиск.**

# Фильтры Блума (2)

**Пусть $x \in \{0,1\}^*$ – некоторое слово в двоичном алфавите. Дайджестом слова *x* называется строка $W(x) = f_1(x),...,f_r(x)$, где $f_j(x)$ – значение псевдослучайной функции *f* с ключом *j*, которое она принимает на слове *x*.**

***Предварительный этап.*** **Пользователь *U* загружает набор из *n* документов на сервер *S* и хочет использовать индекс для поиска по документам. Для этого он должен выполнить следующую последовательность шагов.**

**1. Вначале необходимо получить приемлемые параметры фильтров Блума. Предположим, что фильтр Блума сделан массивом размера *m* битов. Вместо использования *r* независимых хэш-функций используется одна и та же псевдослучайная функция *f* с *r* различными ключами. Ключи выбираются случайно и равновероятно из ключевого пространства.**

**2. Каждому документу последовательно присваивается индекс – целое число из диапазона $[1;n]$.**

**3. Каждый документ сканируется, а именно, для каждого слова *x* в *j*-м документе сначала вычисляется $W(x) = f_1(x),...,f_r(x)$, а затем вставляется $f_{f_1(x)}(j),...,f_{f_r(x)}(j)$ в фильтр Блума для *j*-го документа.**

**4. Каждый документ шифруется и сжимается, используя стандартные алгоритмы. Каждый документ и связанный с ним фильтр Блума помечается присвоенным ему целым числом, после чего пересылается на сервер *S*.**

# Фильтры Блума (3)

***Протокол поиска.*** **Предположим, что пользователь *U* хочет забрать с сервера *S* копии всех документов, которые содержат слово *y*. Тогда он выполняет следующий протокол.**

**1. *U* пересылает *S* сообщение $W(y) = f_1(y),..., f_r(y)$.**

**2. Для каждого *i*-го документа в составе набора из *n* документов *S* вычисляет $f_{f_1(y)}(i),..., f_{f_r(y)}(i)$ и проверяет фильтр Блума *i*-го документа на соответствие.**

**3. *S* возвращает *U* все подошедшие документы.**

***Протокол обновления БД.*** **Для добавления новых документов используется протокол предварительного этапа. При удалении документ и его фильтр Блума просто удаляются с сервера *S*.**

**Изменение содержания существующего документа требует присвоения документу нового уникального номера и перегенерации фильтра Блума, основываясь на этом новом номере. Использование нового номера предотвращает статистический анализ обновлений содержания документов.**

# Схема частично гомоморфного шифрования Пайе (1)

**Схемы конфиденциального выбора используют вспомогательную конструкцию – *схему частично гомоморфного шифрования*.**

**Пусть $(K, E, D)$ – схема открытого шифрования. Чтобы построить схему гомоморфного шифрования, к ней предъявляются следующие требования.**

- **Схема должна быть стойкой как минимум к атакам по выбранным открытым текстам.**
- **Схема должна быть вероятностной (это следствие первого требования).**
- **Схема должна быть *гомоморфной* над абелевой группой *G*.**

**Последнее требование означает следующее. Пусть открытые тексты являются элементами группы *G*, шифртексты – элементами группы $G'$. Тогда $D\left(E(a) \otimes E(b)\right) = a \times b$, где $a, b \in G$, $\times$ – операция в группе *G*, $\otimes$ – операция в группе $G'$. $|G| > 1$. Выберем элемент $g \in G$ и предположим, что $ord(g) = m$, так как в *G* должен быть хотя бы один элемент порядка, большего 1. На практике часто берут в качестве *G* группу целых чисел с операцией сложения. Тогда БД возможно представить как $X = \{x_i\}_{i=1}^{n}$, где $x_i \in \{0, 1, ..., m-1\}$. Как частный случай, БД может интерпретироваться как последовательность битов, т.е. $x_i \in \{0, 1\}$. Гомоморфного шифрования оказывается достаточно, чтобы построить схему конфиденциального выбора информации из БД.**

# Схема частично гомоморфного шифрования Пайе (2)

***Пример*** **схемы гомоморфного шифрования – схема открытого шифрования Пайе (*P. Paillier, 1999*). Пусть** $\mathbf{Z}_{n^2}^* = \{u < n^2 \mid u \equiv 1 \bmod n\}$ **– мультипликативная подгруппа целых чисел, взятых по модулю** $n^2$**, над которой определена функция *L*, такая что для** $\forall u \in \mathbf{Z}_{n^2}^*\ L(u) = (u-1)/n$**. Выберем *p*, *q* – два больших простых числа. Пусть** $n = pq$**. Обозначим через** $B_\alpha \subset \mathbf{Z}_{n^2}^*$ **множество элементов группы, имеющих порядок** $n\alpha$**, *B* – их объединение для** $\alpha = 1,\ldots,\lambda$**, где** $\lambda = \lambda(n) = \text{НОК}(p-1, q-1)$ **– функция Кармайкля.**

**Пусть** $g \in B$ **– случайно выбранный элемент этого множества. Для того, чтобы выбрать его, достаточно проверить выполнение условия** $\text{НОД}\left(L\left(g^\lambda \bmod n^2\right), n\right) = 1$**. Тогда** $(n, g)$ **– открытый ключ,** $(p, q)$ **или** $\lambda$ **– секретный ключ схемы Пайе.**

***Алгоритмы зашифрования и расшифрования схемы открытого шифрования Пайе***

| | |
|---|---|
| **Algorithm** $E_{n,g}(m)$**:**<br>***m***: $m < n$ **– открытый текст;**<br>***r***: $r < n$ **– случайное целое число;**<br>***C***: $C = g^m \cdot r^n \bmod n^2$ **– шифртекст.** | **Algorithm** $D_{p,q}(C)$**:**<br>***C***: $C < n^2$ **– шифртекст;**<br>$m = \dfrac{L\left(C^\lambda \bmod n^2\right)}{L\left(g^\lambda \bmod n^2\right)} \bmod n$ **.** |

**Стойкость схемы Пайе основана на сложности решения задачи распознавания квадратичных вычетов при неизвестных *p* и *q*.**

# Схема конфиденциального выбора информации из БД (1)

**Пусть БД есть последовательность битов (они хранятся в виде открытого текста), $i^*$ – индекс бита БД, к которому хочет получить доступ пользователь *U*. $G, G'$ – группы, из которых выбирается открытый текст и шифртекст, причем группа *G* – мультипликативная, группа $G'$ – аддитивная, $g \in G$ – неединичный элемент группы.**

**Обозначим через *U* пользователя, которому необходимо получить доступ к одной из записей БД, через *P* – провайдера, обеспечивающего сервис хранения данных и доступ к базе данных.**

***Протокол доступа к БД с использованием схемы гомоморфного шифрования* выглядит следующим образом.**

**1. *U* пересылает *P* сообщение $Q = \{q_i\}_{i=1}^{n}$, где $\forall q_i \in G'$: $D_{p,q}(q_i) = g$, если $i = i^*$, и $D_{p,q}(q_i) = 1_G$, если $i \neq i^*$ (так как схема открытого шифрования – вероятностная, то все $q_i$ различны).**

**3. *P* пересылает *U* сообщение $R = \sum_{i=1}^{n} x_i q_i$.**

**4. *U* вычисляет:**

$$D_{p,q}(R) = D_{p,q}\left(\sum_{i=1}^{n} x_i \cdot q_i\right) = \sum_{i=1}^{n} x_i \cdot D_{p,q}(q_i) = x_{i^*} \cdot D_{p,q}\left(q_{i^*}\right) = x_{i^*} \cdot g.$$

**Отсюда $x_{i^*} = 1$ тогда и только тогда, когда $D_{p,q}(R) = g$.**

# Схема конфиденциального выбора информации из БД (2)

**Оценим коммуникационную сложность протокола. Если $k = \log|G'|$, то *U* должен переслать $O(nk)$ битов. Это очень много, пропорционально объему БД.**

**Чтобы снизить коммуникационную сложность, БД лучше организовать в виде квадратной таблицы: $X = \{x_{i,j}\}_{i,j=1}^{\sqrt{n}}$.**

**Пусть $(i^*, j^*)$ – индекс позиции, к которой *U* хочет получить доступ. Тогда протокол будет выглядеть следующим образом.**

**1. *U* пересылает *P* сообщение $Q = \{q_i\}_{i=1}^{\sqrt{n}}$.**

**2. *P* вычисляет $R_j = \sum_{i=1}^{\sqrt{n}} x_{i,j} q_i$ для каждого *j*.**

**3. *P* пересылает *U* сообщение $\{R_j\}_{j=1}^{\sqrt{n}}$.**

**4. *U* вычисляет: $D(R_j) = \{x_{i^*,j}\}_{j=1}^{\sqrt{n}}$, в том числе $x_{i^*,j^*}$.**

**Теперь коммуникационная сложность равна $O(\sqrt{n})$ битов, но и в обратную сторону посылает не одна величина *R*, а $\sqrt{n}$ величин $R_j$.**