Лекция 6

Асиметрични криптосистеми

6.1 Идеята за асиметрична криптосистема

Основна черта на асиметричните криптосистеми е възможността за лесно намиране на метода за дешифриране (дешифриращия ключ) от метода за шифриране (шифриращия ключ). В редица случаи (например, DES) те дори съвпадат. [1] Оттук следва, че шифриращата трансформация (ключът за шифриране) трябва да бъде пазена в тайна, защото публикуването му компрометира криптосистемата. Разбира се, методът на шифриране, формално определя и дешифрирането в математически смисъл, защото двете преобразувания са взаимно-обратни. Но ако се окаже, че намирането на дешифриращата трансформация води до неосъществимо (unfeasible) изчисление, то знанието на шифриращия алгоритъм не компрометира сигурността на системата и тя може да бъде публикувана. Една задача наричаме *изчислително неосъществима*, ако ресурсите, необходими за пресмятането ѝ, измерени вобем използвана памет и необходимо време, надхвърлят всички мислими технологични стойност.[2]

Идеята за асиметрия в шифрирането и дешифриранео възниква в началото на 70-те години и се асоциира с поне две различни групи от изследователи. Дълго време тч се свързваше с работата на Diffie и Hellman [20]. В тази работа дори е предложена конкретна система за обмен на ключове, основана на трудността на намирането н дискретен логаритъм в мултипликативната група на крайно поле. Но в края на 80-те години стана известно, че през 1969 г. J. Ellis, сътрудник на центъра за комуникация на Британското правителство (GCHQ – General Communication Headquarters), открива концепцията за криптография с открит ключ (или несекретно шифриране (n0n-secret encryption) в неговата терминология) като средство за решаване на задачата за обмен на ключове. Ellis, както и Diffie и Hellman, не успял да разработи в детайли работеща криптосистема с открит ключ. Тази задача е решена

[1] Това не е непременно недостатък. Съвпадението на шифриращата с дешифриращата трансформация дава възможност за използване на едно и също оборудване както за шифриране, така и за дешифриране.

[2] Разбира се тази дефиниция, взета от Дифи и Хелман е доста неясна и изпълнена с капани. Да помислим само за квантовата криптография …

през 1973 г. от новия сътрудник на GCHQ, C. Cocks. Той разработва криптосистема, която по същество е еквивалентна на RSA, открита пет години по-късно от R. Rivest, A. Shamir и L. Adleman [42]. През следващата година друг служител на GCHQ, M. Williamson, публикува концепцията за обен на ключове, основана на дискретен логаритъм, която днес е известна като алгоритъм на Diffie-Hellman.

В статията си [20] Diffie и Hellman предлагат концепция за системи от нов вид, в които две партии (хора, устройства) обменят данни *единствено* по публичен канал и използват *единствено* публично известни техники за създаване на сигурна връзка. Те наричат новите системи *криптосистеми с публичен ключ* и ги дефинират като две семейства от алгоритми $\{E_k\}$ и $\{D_K\}$, $k \in \mathcal{K}$, представящи обратими трансформации:

$$E_K : \mathcal{M} \to \mathcal{M}, \quad D_K : \mathcal{M} \to \mathcal{M}$$

на кройно множество в себе си (приемаме, че множествата от откритите текстове и криптотекстовете съвпадат) такива, че

(PK1) за всеки ключ $K \in \mathcal{K}$ EKk е лява обратна на D_K, т.е. $D_K(E_K(m)) = m$ за всяко $m \in \mathcal{M}$.

(PK2) За всяко $K \in \mathcal{K}$ и всяко $m \in \mathcal{M}$.

(PK3) За почти всяко $K \in \mathcal{K}$ е изчислително неосъществимо да се намери алгоритъм D_K^*, за който $D_K^*(E_K(m)) = m$ за почти всички m.

Понякога изискваме и допълнителните условия:

(PK1') за всеки ключ $K \in \mathcal{K}$ E_K е дасна обратна на D_K, т.е. $E_K(D_K(M)) = M$ за всяко $M \in \mathcal{M}$.

(PK3') За почти всяко $K \in \mathcal{K}$ е изчислително неосъществимо да се намери алгоритъм D_K^*, за който $E_K(D_K^*(M)) = M$ за почти всички M.

Да отбележим, че свойства (PK3) и (PK3') не са точно дефинирани. Техният точен смисъл се определя от приловенищщ. Тук приемаме неявно, че алгоритмите E_K и D_K могат да бъдеат ефективно построени при зададено K.

Нека е дадена криптосистема, при която всеки потребител U притежава двойка алгоритми E_U и D_U, и криптосистемата удовлетворява (PK1)–(PK3). Ако A иска да изпрати на B съобщението m в шифриран вид, то тя намира публичния алгоритъм E_B на B, пресмята $c = E_B(m)$ и изпраща c на B. B възстановява m, пресмятайки

$$D_B(c) = D_B(E_B(m)) = m,$$

като второто равенство следва от свойство (PK1). За да е практична системата трябва да удовлетворява (PK2). Накрая свойство (PK3) осигурява сигурността на системата. От това свойство следва, че е допустимо публикуването на E_B без да се компрометира сигурността. Ако B иска да промени тайния си ключ, то той просто генерира нова двойка алгоритми.

Нека сега приемем, че имаме криптосистема, за която са в сила свойствата (PK1'), (PK2) и (PK3'). Такава система може да се използва за създаване на цифрощи подписи. Ако A иска да подпише съобщение m (без да скрива съдържанието му), тя преобразува съобщението с дешифриращия си алгоритъм (с тайния си ключ) $c = D_A(m)$ и изпраща на B двойката (m, c). За да провери подписа B сравнява m със $E_A(c)$. При съвпадение подписът се приема. От свойство (PK3') следва, че A е единственият потребител, който може да пресметне c от m. Така c е подпис за съобщението m.

Да отбележим, че при тази схема всеки потребител може да генерира двойка (m, c), в която c е подпис за m: просто тој избира произволно c и пресмята $m = E_A(c)$. Но вероятността при произволно c да се получи смислено m е пренебржимо малка. Тази вероятност може да се намали още повече, ако наложим определена структура на на m, например полагане на дата и час.

Ако искаме криптосистема, която да осигурява конфиденциалност и цифров подпис, то изискваме свойствата (PK1)–(PK3), (PK1'), (PK3'). Ако A иска да изпрати шифрираното съобщение m, подписано с нейния цифров подпис, то тя извършва следните стъпки. A изпраща на B съобщението

$$c = E_B(D_A(m)),$$

а B възстановява m от c, пресмятайки

$$\begin{aligned} E_A(D_B(c)) &= E_A(D_B(E_B(D_A(m)))) \\ &= E_A(D_A(m)) \\ &= m. \end{aligned}$$

Макар всеки да има достъп до E_B, само B може да възстанови m от c. B запазва $D_B(c)$, което е равно на $(D_B(E_B(D_A(m))))$ като подпис на A за m.

Най-същецвената част в построяването на асиметрична криптосистема се състои в намирането на "лесно" осъществимо преобразуване, което е "трудно" да бъде обърнато. засега ще оставим без пояснение понятията "лесно" и "трудно". [3] Ще илюстрираме идеята за асиметрична криптография върху един пример-играчка.

Пример 6.1. Да резгледаме телефонния указател на голям град. В нашия пример сме използвали телефонния указател на Техническия Университет Мюнхен от 1990 (използвали сме само последните четири цифри). За всяка буква от открития текст избира име от указателя, започващо с тази буква. Телефонният номер на съответния човек е криптотекстът за въпросната буква. Това не е моноалфабетна система, тъй като съществуват много имена започващи със всяка буква и изборът име от указателя се извъшва при допускането, че всички имен аса равновероятни. Така изглежда доста невероятно две еднакви букви, срещащи се на различни места да се шифрират по един и същи начин. Така откритият текст TELEPHONE може да се шифрира по следния начин:

[3] Бихме могли да считаме, че "лесно" е такова преобразувание, което се пресмята чрез полиномиален алгоритъм (от ниска степен, за предпочитане линеен) от размера на входа; а трудно преобразувание е такова, за което не е известен такъв алгоритъм.

T	Thoma	8141
E	Engels, Th.	2027
L	Leibnitz Rechenzentrum	7401
E	Eiselle, Brigitte	8163
P	Preuß	8252
H	Heise	8149
O	Obermeier	2030
N	Nazareth, Dieter	8166
E	Ehler	2348

Полученият криптотекст е

8141 2027 7401 8163 8252 8149 2030 8166 2348

Криптотекстът може лесно да се дешифрира, ако съществува телефонен указател, който е сортиран в нарастващ (намаляващ) ред на телефонните номера. Такъв телефонен указатл може да е достъпен само на авторизиран потребител. Без такъв указател дешифрирането би представлявало значителна трудност.

6.2 Еднопосочни функции

6.2.1 Силни еднопосочни функции

Ще предполагаме, че A и B (Боб) искат да използват криптосистема, в която шифрирането от A и дешифрирането от B са изчислително 'лесни', но задачата за дешифриране (криптанализ) от O (Оскар) е 'трудна'. Разликата в изчислителната трудност между задачите, пред които са изправени A и B, от една страна , и O – от друга, е в основата на съвременната криптография. Такава разлика съществува, ако поставим граница върху изчислителните възможности на O. Реалистично е да допуснем, че същото допускане е в сила за A и B. Така ще приемем, че:

- A, B и O могат да изършват пресмятания с вероятностно полиномиална сложност.

За A и B това означава просто, че трябва да използва вероятностни алгоритми, които са полиномиални по време, за техните шифриращи и дешифриращи трнасформации. Сега ще формализираме идеята, че O трябва да се изправи пред изчислително трудна задача, когато се опитща да дешифрира съобщение без да притежава таjния ключ на B.

Да допуснем, че $\mathtt{P} \neq \mathtt{NP}$ и следователно за никоя NP-трудна задача не съществува полиномиален алгоритъм. Нека A и B използват криптосистема, в която задачата за дешифриране е NP-трудна за O. Въпросът е дали това ще гарантира, че тяхната криптосистема е сигурна. Отговорът е "не" Несъществуването на полиномиален алгоритъм за дадена задача не означава, че той е труден във всички случаи. Задачата може да е лесна върху повечето входове и трудна за нчколко специални входа. Такава крптосистема н може да се счита сигурна. Ще ни е необходимо пончтие за ртудност, което не се основава на поведение в най-лошия случай.

Не е смислено да се иска O да не е никога в състояние да дешифрира каквато и да било криптограма. Ако O просто се опита да отгатне съобщението, то всеки път ще има малка, но все пак ненулева вероятност за това O да отгатме правилно. Така за нас ще е достатъчно да достигнем следното ниво на сигурност.

- *Ако O използва вероятностен полиномиален алгоритъм (повреме), то вероятността той да дешифрира правилно шифротекст $c = E(m)$ от случайно съобщение m е пренебрежима.*

Освен това ще искаме дори O да повтори атаката си полиномиален брой пъти вероятността за успех да бъде малка.

Една функция $\mu : \mathbb{N} \to \mathbb{R}$ наричаме *пренебрежимо малка*, ако за всеки полином p съществува цяло число n_0, за което $\mu(k) < 1/p(n)$ за всички $n \geq n_0$. Така по дефиниция пренебрежимо малка функция е по малка от реципрочната на всеки положителен полином от някакво място нататък. Оттук нататък всички разглеждани полиноми са положителни, т.е. те изпъняват $\mu(n) \geq 1$ за всички цели числа $n \geq 1$. Горната дефиниция се съгласува добре с идеята, че допустими за участниците са само полиномиални пресмятания.

Теорема 6.2. Ако вероятността даден алгоритъм да успее да пресметне дадена изчислителна задача с вход с дължина k е пренебрежимо малка по k, то и вероятността, че той ще успее и след полиномиален брой повторения също е пренебрежимо малка.

(1) *Лесна за пресмятане.* Функцията f може да се пресметне за полиномиално време.

(2) *Трудна за обръщане.* Всеки вероятностен алгоритъм за обръщане на $f(x)$, при зададена случайна стойност $y = f(cx)$ (x е избрано случайно) има пренебрежимо малка вероятност за намиране на прообраз x.

Дефиниция 6.3. Една функция $f : \{0,1\}^* \to \{0,1\}^*$ е *силно-еднопосочна функция*, когато са изпълнени условията

(1) Съществува (детерминистичен) полиномилаен алгоритъм A, който за вход X извежда $f(x)$.

(2) За всеки вероытностен полиномиален (по време) алгоритъм A' и полином $p(.)$ и всички достатъчно големи n:

$$\Pr(A'(f(U_n)), 1^n) \in f^{-1}(f(U_n))) < \frac{1}{p(n)}.$$

Тук U_n е случайна променлива величина равномерно разпределена върху $\{0,1\}^n$. Искаме A' да изведем един (кой да е) обратен на $y = f(x)$. По принцип може да съществуват и повече обратни. Вторият аргумент просто задава дължината на желания изход в унарен запис. Причина е да избегнем обявяването за еднопосочни на функции, които просто драстично скъсяват входа. Така сме гарантирали, че дължината на входа е n. Нека, например $f(x) =$ дължината на двоичното представяне на дължината на низа x. Сега дължината на $f(x)$ е около $\log x$ и очевидно при зададено $y = f(x)$ не можем да изведем низ с $|x|$ символа (да речем $0^{|x|}$) за време, което е полином от $|y|$. Ако f запазва дължината, то вторият аргумент на A' е излишен.

Трудността за обръщане на f може да се интерпретира и като горна граница за вероятността за успех на ефективен алгоритъм за обръщане на f. Входното разпределение на обръщащия алгоритъм се получава като приложим f към равномерно избрано $x \in \{0,1\}^n$. Ако f индуцира пермутация върху $\{0,1\}^n$, то входът на A' е също равномерно разпределен. Но в общия случай f не е непременоо биективна и входното разпределение за A' може да се различава значително от равномерното.

Пример 6.4. Нека алгоритъмът A_1 при вход $(y, 1^n)$ просто избира по случаен начин (равномерно) низ с дължина n. Вероятността за успех на A_1 е равна на вероятността на за колизия на случайната величина $f(U_n)$ (съвпадение на две наблюдения на $f(U_n)$). Ако U'_n е случайна величина равномерно разпределена върху $\{0,1\}^n$, независима от U_n, имаме

$$\begin{aligned}\Pr(A_1(U_n, 1^n) \in f^{-1}(f(U_n))0 &= \Pr(f(U'_n = f(U_n)) \\ &= \sum_y \Pr(f(U_n)) = y)^2 \geq 2^{-n}.\end{aligned}$$

Последното неравенство следва от факта, че $\sum x_i^2 \to \min$, когато всички x_i са равни. Така вероятността за успех на тривиалния алгоритъм е строго положителна. За всяка 1-1 функция вероятността за успех на A_1 е пренебрежимо малка. Накрая, ако f е еднопосочна вероытността за колизия на U_n е пренебрежимо малка.

Пример 6.5. Друг тривиален пример е алгоритъмът A_2, който връща константа за всички възможни входове с една и съща дължинз, т.е $A_2(y, 1^n) = 0^n$. Сега за всяка функция f имаме:

$$\begin{aligned}\Pr(A_2(f(U_n), 1^n) \in f^{-1}(f(U_n))) &= \Pr(f(0^n) = f(U_n)) \\ &= \frac{|f^{-1}(f(0^n))|}{2^n \geq 2^{-n}}.\end{aligned}$$

И тук можем да направим аналогични наблюдения.

Отрицнание на понятието пренебрежимо малка дроб/вероятност е понятието значима дроб/вероятност. Казваме, че функцията $\nu : \mathbb{N} \to \mathbb{R}$ е значима, ако съществува полином $p(.)$ така че за достатъчно големи n е в сила $\nu(n) > \frac{1}{p(n)}$.

6.2.2 Слаби еднопосочни функции

Дефинираните функции в 6.2.1 са еднопосочни с много силно ограничително условие. Всеки ефективен алгоритъм за обръщането им има пренебрежимо малка вероятност за успех. Сега ще дадем друга, по-слаба дефиниция, която изисква само всеки ефективен алгоритъм за обръщането им да не успява със значима вероятност.

Дефиниция 6.6. (*слаби еднопосочни функции*) Една функция $f : \{0,1\}^n \to \{0,1\}^n$ наричаме слаба еднопосочна функция, ако са в сила условията:

(1) Съществува (детерминистичен) полиномилаен алгоритъм A, който за вход X извежда $f(x)$. ((както при силно-еднопосочните функции)

(2) Съществува полином $p(.)$, такъв че за всеки веоятностен полиномиален алгоритъм A' и за всички достатъчно големи n:

$$\Pr(A'(f(U_n)), 1^n) \notin f^{-1}(f(U_n))) > \frac{1}{p(n)}.$$

6.2.3 Кандидати за еднопосочни функции

Към настоящия момент обект на разглждане са няколко кандидата за еднописични функции. Разбира се не е известно дали тези функции наистина са еднопосочни.

А. Разлагане на прости множители. Да разгледаме функция $f_{\text{мулт}}$, която получава като вход два двоични низа и връща на изхода двоичното представяне на произведението им : $f_{\text{мулт}}(x, y) = x \cdot y$. Обратната задача е задачата за разлагане на прости множители. Ще отбележим, че към настоящия момент най-добрите известни алгоритми за разлагане на прости множители са със субекспоненциална сложност (по щреме) ин нашето очакване е, че тя е "трудна". Това се формулира като т.нар. factoring assumption.

Factoring assumption. *За всеки положителен полином, за всеки вероятностен алгоритъм и за всички достатъчно големи k е в сила*

$$\Pr(A(n) = ab) \leq \frac{1}{p(k)},$$

където $n = ab$, а a и b са случайни k-битови прости числа.

Като използваме теоремата за броя на простите числа Prime Number Theorem (PNT)

$$\lim_{n\to\infty} \frac{\pi(n)\log n}{n} = 1,$$

можем да получим, че $f_{\text{мулт}}$ е слабо еднопосочна функция.

Нека P_k е множеството на k-битовите прости числа. От PNT лесно следва, че вероятността случайно избрано k-битово цяло число да е просто е по-голяма от $1/k$ (да се докаже). Оттук вероытността две независимо избрани случайни числа да са прости е по-голяма от $1/k^2$. За значима част от входове за $f_{\text{мулт}}$ задачата за обръщане ще е трудна съгласно Factoring Assumption. Нека A е вероытностен алгоритъм с полиномиална сложност с вход $n = ab$. $a, b \in P_k$ (избрани случайно равномерно). Вероятността за неуспех на Factoring Assumption е

$$\Pr(A \text{ не успява да обърне } n) \geq 1 - \frac{1}{p(k)} \geq \frac{1}{2}$$

за достатъчно големи k. Така за достатъчно големи k и произволни k битови цели числа числа a и b, и $n = ab$ имаме

$$\begin{aligned}
\Pr(A \text{ не успява да обърне } n) &\geq \\
&\geq \Pr(A \text{ не успява да обърне } n) \mid a, b \in P_k) \cdot \Pr(a, b \in P_k) \\
&\geq \frac{1}{2} \cdot \frac{1}{2^k} = \frac{1}{2k^2},
\end{aligned}$$

т.е. вероятността е значима и $f_{\text{мулт}}$ е слабо еднопосочна.

Ако разгледаме $f_{\text{мулт}}$ само върху простите числа

$$f_{\text{pmult}}(x, y) = xy, x, y \in P_k,$$

то тогава $f_{\text{пмулт}}$ е силно еднопосочна (по дефиниция).

B. Пресмятане на дискретен логаритъм Нека p е просто число, g е примитивен елемент по модул p (пораждащ на $\mathbb{Z}_p^*$), а $x \in \{0, \ldots, p-2\}$. Дефинираме

$$\mathtt{dexp}(p, g, x) = (p, g, g^x \mod p).$$

Стойностите на функцията $\mathtt{dexp}(p, g, x)$ се пресмята лесно, тъй като повдигането на степен по модул може да се изпълни за полиномиално време (да се докаже!). Дефинираме обратната функция за `dexp` чрез

$$\mathtt{dlog}(p, g, y) = (p, g, x),$$

където $y = g^x \pmod p$. Пресмятането на `dlog` е известно като задача за намиране на дискретен логаритъм. За нея се счита, че е изключително трудна. Към настоящия момент най-добрият алгоритъм има очаквано време за изпълнение

$$O(exp(c(\ln p)^{1/3}(\ln \ln p)^{2/3})).$$

Както и по-горе, естествено е да се приеме следното допускане за трудността за намиране на дискретен логаритъм.

Discrete Log Assumption. *За всеки положителен полином $q(.)$, за всеки вероятностен алгоритъм A и за всички достатъчно големи k е в сила*

$$\Pr(A(p, g, y) = \mathtt{dlog}(p, g, y)) \leq \frac{1}{q(k)},$$

където p е случайно k-битово просто число g е случаен примитивен елемент по модул p, а x е случаен елемент от $\{0, 1, \ldots, p-2\}$.

C. Декодиране на линейни кодове Една от важните задачи в теория на кодирането е построяването на ефективен алгоритъм за случайни линейни кодове. Специален интерес представляват кодове с постянна скорост ($R = k/n$) и постоянно относително минимално разстояние ($\delta = d/n$). Границата на Варшамов-Джилбърт за линейни кодове гарантира съществуването кодове със скорост $r < 1 - h(\delta)$, където $h(p) = -p \log p - (1-p) \log(1-p)$, $p < 1/2$, и $h(p) = 1$ – в останалите случаи ($h(p)$ е модификация на обичайната ентропия).

По подобен начин, ако за някое $\varepsilon > 0$ имаме

$$\frac{k}{n} < 1 - h\left(\frac{(1+\varepsilon)d}{n}\right),$$

то почти всички $k \times n$ двоични матрици ще са пораждащи за $[n, k, d]$-кодове. Да разгледаме константи $\kappa, \delta, \varepsilon > 0$, за които $\kappa < H((1 = \varepsilon)\delta)$. Сега функцията $f_{\text{цоде}}$ е кандидат за еднопосочна функция:

$$f_{\text{code}}(C, x, i) := (C, xC + e(i)).$$

Тук C е $\kappa n \times n$- двоична матрица, x е κn-мерен двоичен вектор, а i е индекс (номер) на n-мерен двоичен вектор с тегло на Хеминг най-много $\delta n/2$. Ясно е, че $f_{\text{цоде}}$ се пресмята в полиномиално време. Ефективен алгоритъм за декодиране би дал ефективен алгоритъм за обръщане на $f_{\text{цоде}}$.

D. Subset sum problem Нека $|x_1| = \ldots = |x_n| = n$ (числа с дължина n в двоичен запис) и нека $I \subset \{1, \ldots, n\}$. Дефинираме

$$f_{\text{ssum}}(x_1, \ldots, x_n, I) = (x_1, \ldots, x_n, \sum_I x_i).$$

Очевидно функцията $f_{\text{ссум}}$ се пресмята за полиномиално време, но пресмятането на обратната ѝ е NP-пълна. разбира се, този факт не е доказателство, че $f_{\text{ссум}}$ е еднопосочна. От друга страна, фактът, че обръщането е лесно в някои специални случаи (наприме, при налична скрита структура или ниска плътност) не отхвърля този кандидат. Хипотезата, ‘е $f_{\text{ссум}}$ е еднопосочна се основава на неуспеха на известните алгоритми да се справят със случайни индивидуални задачи с висока плътност (т.е. такива, в които дължината на елементите е приблизително равна на броя им).

6.2.4 Еквивалентност на силни и слаби еднопосочни функции

Теорема 6.7. *Слаби еднопосочни функции съществуват тогава и само тогава, когато съществуват силни еднопосочни функции.*

Доказателство. Ще даем само идея за доказателство на този резултат. Пълно доказателство може да бъде намерено в монографията на O. Goldreich, Foundations of Cryptography.

В едната посока твърдениыето е тривиално: всяка силна еднопосочна функция е и слаба еднопосочна функция.

Нека $f : \{0, 1\}^* \to \{0, 1\}^*$ е слаба еднопосочна функция. Целта ни е да построим от f силна еднопосочна функция. По дефиниция опонентът не успява да обърне f за значима част от входовете. Ние ще построим такава нова функция, обръщането на която ще означава, че опонентът може да обърне f за голям брой случайни стойности.

Нека $q(.)$ е положителният полином, асоцииран с f от дефиницията на слаба еднопосочна функция. дефинираме $g : \{0, 1\}^* \to \{0, 1\}^*$ чрез

$$g(x_1 x_2 \ldots x_m) = f(x_1) f(x_2) \ldots f(x_m),$$

където $m = nq(n)$ и всяко x_i е с дължина n.

За да обърне g, опонентът трябва да обърне f за $nq(n)$ стойности $f(x_i)$. Тъй като вероытноста за неуспех при обръщане на едно $f(x_i)$ е поне $1/q(n)$, то вероятността за успех при обръщане на g се задава с

$$
\begin{aligned}
\Pr(\text{Invert } g(x_1 \dots x_m) &= \Pr(\text{Invert } F(x_1), \dots, f(x_m)) \\
&\leq \left(1 - \frac{1}{q(n)}\right)^{nq(n)} \\
&\cong e^{-n}.
\end{aligned}
$$

Тук вероятността опонентът да да пресметне обратната стойност за случаен вход на g пренебрежимо малка и така g е силна еднопосочна функция. □

Забележка 6.8. В горното доказателство неявно е направено допускането, че за да обърне g опонентът трыбва да пресметне обратната стойност за всяко $f(x_i)$, което не е задължително.