Физически неклонируемые функции в криптографии

В. С. Бельский, И. В. Чижов, А. А. Чичаева, В. А. Шишкин

Аннотация— **Физически неклонируемая функция – это аппаратное устройство, экземпляры которого имеют ряд уникальных параметров и характеристик, т.е. в силу особенностей физического процесса, используемого в производстве, невозможно создать два экземпляра, имеющих идентичные значения этих характеристик. Иногда можно считать, что эти характеристики принимают случайные значения. Эта своего рода физическая случайность может быть использована в различных криптографических протоколах и механизмах. Благодаря своей экономичности физически неклонируемые функции являются перспективными для использования в устройствах с ограниченными ресурсами, таких как RFID метки. А таких устройств с каждым днем становится все больше и больше.**

В работе исследуются возможности применения физически неклонируемых функций в криптографических протоколах для решения следующих задач: выработка случайных значений, идентификация и аутентификация объектов.

Ключевой особенностью использования физически неклонируемых функций для генерации случайных параметров является отсутствие необходимости хранить полученные величины в памяти, поскольку их можно каждый раз заново воспроизводить. Это является большим преимуществом, так как параметры протоколов обычно необходимо хранить именно в защищенной памяти, которая является дорогим ресурсом. Однако важно понимать, что при измерении характеристик устройств возникают погрешности, которые приводят к необходимости реализации дополнительных механизмов, исправляющих ошибки. В работе описаны основные конструкции, используемые для этих целей.

На сегодняшний день предложено множество протоколов аутентификации на основе физически неклонируемых функций. Их можно разделить на два класса: протоколы парольной аутентификации с генерацией ключей на основе физически неклонируемых функций и протоколы аутентификации на основе запросов и ответов физически неклонируемых функций. В статье рассмотрены существующие протоколы аутентификации, их преимущества и недостатки.

Помимо криптографических протоколов в работе рассматривается возможность создания математической модели физически неклонируемой функции. Современные методы машинного обучения позволяют математически «клонировать» экземпляры устройств. Этот факт является существенным недостатком физически неклонируемых функций.

На основе рассмотренной информации можно сделать вывод, что физически неклонируемые функции являются перспективными для использования в устройствах с ограниченными ресурсами. Вместе с тем большинство предложенных на текущий момент конструкций имеют ряд эксплуатационных недостатков и уязвимостей к атакам на основе методов машинного обучения, что свидетельствует о преждевременности рассмотрения физически неклонируемых функций в качестве структурного узла криптографических механизмов и протоколов.

Ключевые слова— **Физически неклонируемые функции, Аутентификация устройств.**

I. Введение

Идея физически неклонируемых функций (ФНФ) была впервые сформулирована Р. Паппу в 2002 году в работе [1], в которой он представил концепцию «физически однонаправленных функций». Вскоре после этого Б. Гассенд и др. [2] предложили новую конструкцию кремниевой ФНФ и определили её как «физически случайную функцию». Несмотря на то что эти два определения были сформулированы исторически раньше, в настоящее время в основном употребляется понятие «физически неклонируемой функции» (ФНФ, от англ. Physical Unclonable Functions, PUF).

В настоящее время отсутствует однозначное определение ФНФ. Одно из широко используемых на практике определений ФНФ было предложено П. Туилсом [3]. ФНФ, по его мнению, – это физические системы (устройства), неотъемлемым свойством которых является неклонируемость некоторых их функций, свойств, характеристик либо параметров. Неклонируемость подразумевает, что производственный процесс не позволяет создать два устройства, обладающих одинаковыми физическими характеристиками. Реальные физические устройства обычно обладают указанным свойством, поскольку они состоят из множества компонентов, параметры которых в процессе создания физической системы принимают случайные значения. Среди окружающих нас материальных объектов сложно найти два абсолютно одинаковых. Даже в серийном производстве каждый объект получается уникальным за счет погрешностей и

Статья получена 9 августа 2020
В. С. Бельский, Лаборатория криптографии АО НПК «Криптонит», (e-mail: v.belsky@kryptonite.ru).
И. В. Чижов, Лаборатория криптографии АО НПК «Криптонит», МГУ имени М.В. Ломоносова, Федеральный исследовательский центр «Информатика и управление» РАН (e-mail: i.chizhov@kryptonite.ru),
А. А. Чичаева, Лаборатория криптографии АО НПК «Криптонит», МГУ имени М.В. Ломоносова (e-mail: a.chichaeva@kryptonite.ru).
В. А. Шишкин, Лаборатория криптографии АО НПК «Криптонит», (e-mail: v.shishkin@kryptonite.ru).

случайностей. Например, при изготовлении интегральных схем точные значения пороговых напряжений, задержек распространения сигналов, частоты работы компонентов и т. п. невозможно предугадать заранее.

Эти особенности каждого отдельного объекта можно регистрировать и использовать как уникальный идентификатор, своеобразный «отпечаток пальца». Что приводит к идеи использовать ФНФ в качестве аутентифицирующего фактора сетевого устройства, на котором она установлено. Фактор ФНФ может быть полезен в мобильных устройствах Интернета вещей, компонентах автоматизированных систем управления технологическим процессом и подобных им устройствах, в которых, по разным причинам, невозможно для аутентификации использовать полноценные средства криптографической защиты.

Однако при применении ФНФ в протоколах аутентификации возникают сложности. Обычно для осуществления аутентификации на вход экземпляру ФНФ подаётся запрос, который переводит его в определенное состояние, а после, с помощью процедуры измерения, получают некоторую величину (ответ), характеризующую состояние экземпляра. На этот ответ влияют как внутренние параметры экземпляра, полученные в результате производственной случайности, так и внешние параметры (температура окружающей среды, напряжение питания и другие). Поэтому ответ экземпляра ФНФ на один и тот же запрос может отличаться, что затрудняет его использование для аутентификации.

В качестве наглядного примера ФНФ можно привести следующее оптическое устройство. Если в расплавленное стекло добавить пузырьки воздуха, а после остудить эту массу и разрезать на одинаковые бруски, то пузырьки воздуха внутри каждого бруска будут распределены по-разному. Эти различия можно зафиксировать, отправив на брусок пучок лазерного излучения (запрос) и получив на выходе уникальную интерференционную картину пучков излучения после преломления (ответ). Эта интерференционная картина и будет уникальным, невоспроизводимы идентификатором бруска стекла.

ФНФ являются относительно новым объектом исследований, но имеющим перспективы использования в современных информационных технологиях. Они являются экономичными, используют в работе небольшое количество ресурсов и поэтому легко могут найти себе место в концепции Интернета вещей.

В настоящее время некоторые производители начали использовать ФНФ в своих устройствах. Например, Xilinx и Altera используют ФНФ в качестве встроенного неклонируемого идентификатора программируемых логических интегральных схем и для генерации криптографических ключей в них. Например, в Zynq UltraScale + MPSoC [4] реализованы ФНФ на базе кольцевых генераторов.

Ещё одним примером применения ФНФ на практике является генерация корневых ключей (мастер-ключей) в устройства Интернета вещей. Корневой ключ устройства генерируется из ФНФ. Из этого корневого ключа ФНФ могут быть выработаны производные ключи. Когда возникает необходимость расшифровать данные, ключ просто воспроизводится из экземпляра ФНФ, тем самым он не хранится в памяти постоянно, а вырабатывается в процессе использования.

В этой статье рассматриваются ФНФ, их свойства и возможности применения в криптографических протоколах. Выделяются преимущества и недостатки этой концепции и возможные направления развития.

II. Определение физически неклонируемой функции

A. Инфраструктура классов ФНФ

Следуя работе [5], дадим основные определения.

Для начала введём понятие *класса ФНФ*, обозначаемого как **P**, который описывает конкретный тип конструкций ФНФ. Класс ФНФ **P** задаётся вероятностным алгоритмом $Create$. На вход алгоритма подаётся строка r^c, выбираемая случайно и равновероятно из множества $\{0,1\}^*$ всех двоичных последовательностей конечной длины. Выходом алгоритма является экземпляр puf_{r^c} (конкретная реализация) ФНФ. Функция $\mathbf{P}.Create$ на практике соответствует некоторому технологическому процессу, но его можно рассматривать как вероятностный алгоритм.

$$\mathbf{P} \equiv \{puf_{r^c} \leftarrow \mathbf{P}.Create(r^c): r^c \xleftarrow{\$} \{0,1\}^*\}$$

где знак $\xleftarrow{\$}$ означает, что значение r^c случайно и равновероятно выбирается из множества $\{0,1\}^*$.

Процедуру создания случайного экземпляра ФНФ $puf_{r^c} \leftarrow \mathbf{P}.Create(r^c)$ в дальнейшем будем обозначать как $PUF \leftarrow \mathbf{P}.Create$, или ещё короче $PUF \leftarrow \mathbf{P}$.

puf_{r^c} – это физическое устройство, имеющее внутреннее состояние, определяемое физическими параметрами. Значения параметров устанавливаются с помощью внешнего входа (запроса). Экземпляр класса puf_{r^c} можно рассматривать как вероятностный алгоритм, которому на вход подаётся запрос x. Множество всех возможных запросов x, которые можно подать на вход экземплярам класса ФНФ **P**, обозначается как $X_{\boldsymbol{P}}$. На выходе этот вероятностный алгоритм выдаёт число, характеризующее состояние экземпляра. Для многих классов ФНФ на ответы экземпляров также влияют внешние физические параметры, например, температура окружающее среды, уровень напряжения питания и т. д., поэтому помимо запроса puf_{r^c} принимает на вход параметр α, который определяет условия измерения. Например, $\alpha = (T_{env} = 80\ ℃)$ означает, что эксперимент проводился при температуре окружающей среды 80 ℃. Случайность можно выразить явным образом в виде отдельного входа:

$$puf_{r^c}\left(x, r^E \xleftarrow{\$} \{0,1\}^*, \alpha\right)$$

$puf_{r^c}(x, r^E, \alpha)$ на практике соответствует физическому эксперименту, результатом которого является число (ответ), соответствующее состоянию экземпляра ФНФ при запросе x.

В дальнейшем будем использовать укороченные обозначения. Ответ конкретного экземпляра ФНФ puf_{r^C} на запрос x: $y_{r^C}^{\mathrm{r}^E}(x) \leftarrow puf_{r^C}\left(x, \mathrm{r}^E \overset{\$}{\leftarrow} \{0,1\}^*, \alpha\right)$ обозначим как $Y_{r^C}(x) \leftarrow puf_{r^C}(x)$. Ответ случайного экземпляра ФНФ на запрос x обозначим $Y(x) \leftarrow PUF(x)$. Отметим, что $Y_{r^C}(x)$ и $Y(x)$ – это случайные величины.

Множество всех возможных ответов, которые может генерировать экземпляр ФНФ класса **P**, обозначается как $Y_{\boldsymbol{P}}$. Таким образом ответ экземпляра ФНФ puf_{r^C} на запрос x является случайной величиной $Y_{r^C}(x)$, заданной на множестве $Y_{\boldsymbol{P}}$.

B. Свойства классов ФНФ

Для использования в криптографических протоколах ФНФ должны обладать определенными свойствами. Далее рассмотрим только ключевые из них. В этом разделе рассматриваются неформальные определения этих свойств. Такой подход выбран для того, чтобы передать суть свойств, не нагромождая текст. Формальные определения можно найти, например, в работе [5].

Определение 1. Класс ФНФ **P** называется *конструируемым*, если его алгоритм $Create$ легко выполняется, и тем самым получается случайный экземпляр ФНФ $puf_{r^C} \leftarrow \mathbf{P}.Create(r^C \overset{\$}{\leftarrow} \{0,1\}^*)$.

Нет смысла обсуждать ФНФ, которые невозможно эффективно реализовать, т. е. сконструировать. Слово «легко» в определении зависит от контекста. Поскольку ФНФ являются физическими объектами, то, по крайней мере, необходимо, чтобы существовал технологический процесс их производства. С более практической точки зрения слово «легко» относится к стоимости производства экземпляра определенного класса ФНФ. Свойство конструируемости не противоречит свойству физической неклонируемости, определенному ниже, так как требует возможность создания произвольного экземпляр ФНФ, в то время как неклонируемость подразумевает сложность воспроизведения заданного экземпляр ФНФ.

Определение 2. Класс ФНФ **P** называется *оцениваемым*, если он конструируемый, и для любого случайного экземпляра ФНФ $puf \in \mathbf{P}$ и любого случайного вызова $x \in X_{\boldsymbol{P}}$ легко получить ответ $y \leftarrow puf\left(x, r^E \overset{\$}{\leftarrow} \{0,1\}^*\right)$.

Слово «легко» в определении снова зависит от контекста. С теоретической точки подразумевается полиномиальная выполнимость. Однако на практике «легко» означает, что можно получить ответ при заданных ограничениях на время, площадь, мощность, энергию и затраты.

Определение 3. Класс ФНФ **P** называется воспроизводимым, если каждый экземпляр ФНФ класса **P** на один и тот же запрос выдаёт ответы близкие между собой по метрике.

При исследовании класса на воспроизводимость рассматриваются ответы на случайные вызовы случайных экземпляров ФНФ.

Основным свойством ФНФ, связанным с безопасностью, является уникальность.

Определение 4. Класс ФНФ **P** является уникальным, если он оцениваемый, и с большой вероятностью ответы на один и тот же запрос разных экземпляров ФНФ сильно отличаются в некоторой рассматриваемой метрике.

Если класс ФНФ демонстрирует воспроизводимость и уникальность, то его экземпляры могут быть идентифицированы на основе их ответов.

Определение 5. Класс ФНФ **P** называется *идентифицируемым*, если он воспроизводим и уникален.

Идентифицируемость выражает тот факт, что ответы (на один и тот же запрос), поступающие от одного экземпляра ФНФ, более похожи, чем ответы разных экземпляров.

Если предположить, что противник контролирует процедуру создания экземпляра класса ФНФ, т. е. может влиять на условия, параметры и источники случайности $\mathbf{P}.Create$, то злоумышленник может создать два похожих экземпляра ФНФ, используя влияние на процесс их производства. Чтобы избежать этого, нужно быть уверенным в том, что экземпляры остаются уникальными даже при наличии такого противника.

Определение 6. Класс ФНФ **P** является *физически неклонируемым*, если он оцениваемый, и сложно повлиять на процедуру создания экземпляра класса $\mathbf{P}.Create$ таким образом, чтобы создать два различных экземпляра ФНФ, у которых с достаточно высокой вероятностью близки ответы.

В этом случае фраза «сложно повлиять» означает физические и технические сложности (или невозможность) создания такой пары экземпляров. Эти трудности необходимо оценивать с точки зрения технических возможностей противника, которые зависят от его опыта и располагаемого бюджета на покупку оборудования.

Класс устройств, который демонстрирует физическую неклонируемость, обладает преимуществом безопасности, заключающимся в том, что даже производитель устройств этого класса не имеет возможности нарушить свойство уникальности.

Свойства, описанные выше, являются определяющими для классов ФНФ, то есть отличают их от классов произвольных физических устройств, поэтому можно привести следующее определение класса физически неклонируемых функций:

Определение 7. Класс физических объектов, предоставляющих возможность подавать им запросы и измерять ответы на них, называется классом физически неклонируемых функций, если он является идентифицируемым и физически неклонируемым.

В некоторой литературе классы ФНФ разделяют на сильные и слабые. Эта классификация строится на количестве возможных пар запрос-ответ. По сути, класс ФНФ называется сильным, если даже после предоставления злоумышленнику доступа к экземпляру ФНФ в течение продолжительного периода времени, все ещё можно столкнуться с запросом, на который с высокой вероятностью противник не знает ответ. Впервые эта классификация была предложена в работе [6], а после уточнена в работе У. Рюрмайра и

др. [7], в которой можно найти формальное определение сильным и слабым ФНФ.

Создание практической сильной ФНФ, удовлетворяющей определению из [7], оказывается трудной задачей, и на сегодняшний день, вопрос о том, возможно ли это на самом деле, остаётся открытой технологической проблемой. Поэтому часто в литературе под сильными ФНФ подразумеваются устройства, имеющие большое количество (экспоненциальное относительно некоторого параметра безопасности) уникальных пар запрос-ответ. Остальные классы называются слабыми.

III. Классы ФНФ, реализуемые в цифровых устройствах

Классы ФНФ были предложены на основе широкого спектра технологий и материалов, таких как стекло, пластик, бумага, электронные компоненты и кремниевые интегральные схемы (ИС). Чаще всего в цифровых устройствах используются кремниевые ФНФ, основанные на случайных вариациях, происходящих в процессе производства кремниевых чипов.

В этом разделе будут описаны базовые типы классов кремниевых ФНФ.

A. ФНФ типа арбитр

ФНФ типа арбитр (Arbiter PUF) [8,9] являются одной из наиболее известных и хорошо изученных конструкций. Они основаны на измерении случайных вариаций задержек сигналов в цифровых устройствах. Идея состоит в том, чтобы привнести состояние гонки между двумя путями микросхемы. Оба пути заканчиваются элементом-арбитром, который определяет какой из путей был быстрее и выдает соответствующее бинарное значение.

Для реализации ФНФ этого типа на цифровом устройстве изготавливаются два топологически и функционально идентичных пути. Очевидно, что оба пути будут иметь близкие значения величин распространения по ним сигналов, однако они незначительно изменяются от устройства к устройству из-за наличия технологических вариаций во время изготовления цифровых устройств. Процесс измерения ответа будет заключаться в одновременной подаче на входы обоих путей сигнала и определении, который из них появится на выходе быстрее. Симметричные пути задержки изготавливаются таким образом, что одновременно из большого множества пар путей выбирается одна пара. Далее для выбранной пары путей определяется, какой из них является более быстрым.

В качестве примера можно привести, наиболее распространенную схему ФНФ типа арбитр (Рисунок 1) [9]. Эта схема строится с использованием n последовательно подключенных пар мультиплексоров с двумя входами. (Мультиплексор — логическая схема, имеющая один выход y, 2^n информационных входов $x_0, x_1, \dots, x_{2^n-1}$ и n адресных входов $s_0, s_1, \dots, s_{n-1}$. На выходе мультиплексор выдаёт значение информационного входа x_i, где i — число, которое кодируется адресными входами $s_0, s_1, \dots, s_{n-1}$). Адресные входы обоих мультиплексоров каждой пары объединяются и используются для задания значения запроса. В качестве запроса используется n-разрядный вектор $C_i = c_0 c_1 \dots c_{n-1}$ где $c_j \in \{0,1\}$, $j \in \{0,1,2,\dots,n-1\}$. Запрос C_i в схеме ФНФ типа арбитр формирует два пути таким образом, что если $c_j = 0$, то для построения первого пути используется верхний мультиплексор MUX_j, а для второго – нижний MUX_j, а если $c_j = 1$, то мультиплексоры меняются местами. Каждая пара путей имеет общий адресный вход. Очевидно, что количество пар путей с увеличением n растет экспоненциально и равняется 2^n. Для конкретного запроса C_i генерируется ответ $Y_i \in \{0,1\}$, как результат эксперимента по определению, какой из путей выбранной запросом C_i пары − первый или второй – быстрее. Например, если первый, то принимается $Y_i = 1$, а если второй – $Y_i = 0$. Очевидно, что в силу симметричности путей, на этапе изготовления невозможно предсказать и запрограммировать, какой из путей имеет меньшую задержку. Таким образом, ФНФ типа арбитр позволяет получить однобитный ответ Y_i на n-битный запрос C_i.

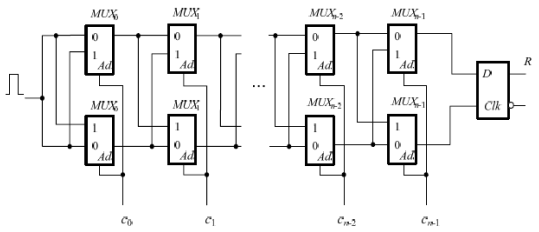

Рисунок 1. ФНФ типа арбитр на основе мультиплексоров

Главной проблемой ФНФ типа арбитр является то, что их можно смоделировать с использованием технологии машинного обучения. В настоящее время были представлены многочисленные результаты, которые успешно применяют различные методы машинного обучения для моделирования поведения ФНФ типа арбитр [10–12].

B. ФНФ на базе кольцевых генераторов

Еще одним типом ФНФ на основе задержек являются ФНФ на базе кольцевых генераторов (Рисунок 2) (Ring Oscillator PUF, кольцевая ФНФ). Кольцевой генератор представляет собой колебательный контур, состоящий из нечетного числа инверторов (устройств, преобразующих постоянный ток в переменный), который служит для генерации последовательности импульсов. Частота импульсной последовательности, полученной на выходе генератора, определяется величиной задержки на элементах генератора. В силу вариаций задержек сигнала элементов генератора два идентичных по топологии и функциональности кольцевых генератора имеют отличающиеся частоты выходных импульсных сигналов. Различие частот сигналов является основой для формирования однобитного ответа. Действительно, две пары кольцевых генераторов на одном либо разных кристаллах будут иметь произвольное соотношение частот и уникально характеризовать полученную пару, либо соответственно кристалл.

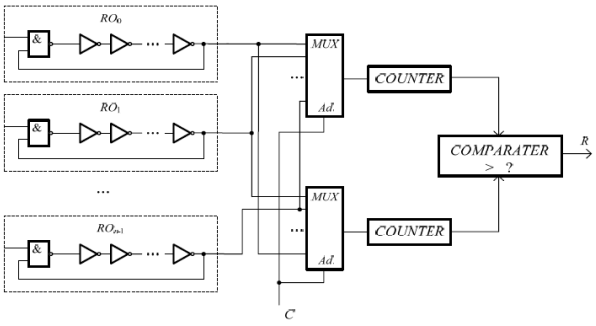

Рисунок 2. ФНФ на базе кольцевых генераторов

Первая кольцевая ФНФ была предложена в [2,13]. Частота одинаково реализованных кольцевых генераторов меняется в зависимости от устройства из-за производственных изменений величины задержки цифровых компонентов генераторов, что позволяет использовать их при построении ФНФ. Но было замечено, что результирующая частота в значительной степени зависит от температуры и внешнего напряжения. Это приводит к тому, что ответ ФНФ зависит не только от конкретного устройства, но и от условий окружающей среды, поэтому устройство ведёт себя непредсказуемо, и будет давать сильно отличающиеся ответы на один и тот же запрос. В таком случае из этих ответов невозможно будет извлечь один и тот же ключ, необходимый для криптографических протоколов. Чтобы противостоять влиянию окружающей среды, предлагается метод постобработки, называемый компенсированным измерением (compensated measuring). Принцип, лежащий в основе этого метода, заключается в рассмотрении отношения частот двух кольцевых генераторов, реализованных на одном устройстве. Так как изменения окружающей среды будут влиять на обе частоты примерно одинаково, то их отношение будет достаточно стабильным и может использоваться в качестве ответа ФНФ. При применении такого метода к кольцевой ФНФ достигается хорошая воспроизводимость ответов.

C. ФНФ на основе элементов памяти

Принцип работы ФНФ на базе устройства, реализующего статическую память с произвольным доступом (SRAM, static random access memory) основан на случайности состояния ячеек памяти при включении питания.

Статическая память с произвольным доступом широко используется в вычислительной технике для хранения данных. Запоминающий элемент SRAM состоит из четырех транзисторов, реализующих два инвертора (логические элементы с отрицанием) с перекрестными обратными связями. Примером ячейки SRAM может служить RS-триггер (Рисунок 3) (триггер, который сохраняет своё предыдущее состояние при неактивном состоянии обоих входов и изменяет своё состояние при подаче на один из его входов сигнала), реализованный на двух логических элементах 2И-НЕ, реализующих штрих Шеффера. RS-триггер получил название по имени своих входов. Вход S (Set — установить англ.) позволяет устанавливать выход триггера Q (Quit — выход англ.) в единичное состояние (записывать единицу). Вход R (Reset — сбросить англ.) позволяет сбрасывать выход триггера Q в нулевое состояние (записывать ноль).

Ячейка SRAM всегда находится в одном из двух состояний (0 или 1), что, в свою очередь, позволяет использовать ее для хранения одного бита информации. Но при этом SRAM энергозависима, поэтому состояние ячеек теряется вскоре после отключения питания, а при включении питающего напряжения все ячейки SRAM устанавливаются в одно из двух возможных состояний – 0 или 1; причем в силу симметричного строения ячеек заранее неизвестно, какое состояние примет ячейка – 0 или 1. Это состояние случайно и определяется множеством факторов.

Экспериментально подтверждено, что большинство ячеек SRAM при включении питающего напряжения преимущественно переходят в одно предпочтительное состояние. Причиной этого является то, что каждая ячейка, представляющая собой RS-триггер, в силу особенностей технологии изготовления имеет множество несимметричных элементов. К таким элементам можно отнести длины соединительных проводников, их геометрические размеры, неоднородность физических и химических свойств кремния и др. В работе [6] было показано, что только для части ячеек SRAM их состояние после включения питающего напряжения является действительно случайным и зачастую приближается к равномерному распределению. Остальные ячейки принимают стабильное значение.

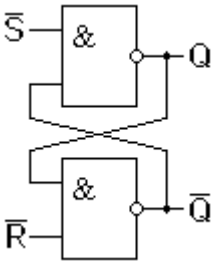

Рисунок 3 - RS-триггер

D. Комбинации ответов экземпляров ФНФ

Ответы экземпляров ФНФ обычно используются как некий источник случайности, но оказалось, что для большинства классов ФНФ методы машинного обучения позволяют построить математическую модель экземпляра ФНФ по небольшому числу известных пар запрос-ответ. Для увеличения случайности и усложнения процесса построения модели ФНФ было предложено комбинировать ответы нескольких экземпляров ФНФ в один. Это можно сделать разными способами. Приведём несколько примеров.

***k*-XOR ФНФ.** Чаще всего множеством возможных ответов ФНФ являются битовые векторы фиксированной длины m. Самым простым примером комбинации таких ФНФ служит сложение по модулю два (XOR) ответов нескольких экземпляров ФНФ. В этом случае ответы строятся следующим образом: создаются k независимых экземпляров ФНФ одного класса. Запрос x_i подаётся на вход каждому из них, а затем их ответы $y_i^1, \dots, y_i^k$ складываются по модулю два, и получается общий ответ y_i. Чем больше число k, тем сложнее предсказать результат XOR ФНФ, но при этом

увеличивается и погрешность ответа. На практике увеличение шума даёт ограничение на максимальное количество экземпляров ФНФ, участвующих в комбинировании.

Изначально такая схема была предложена для ФНФ типа арбитр, но потом стали комбинировать ответы и других классов, например ФНФ на базе кольцевых генераторов.

В настоящее время были найдены эффективные атаки [14] на такой способ комбинирования ФНФ.

Ещё одним способом комбинации ответов ФНФ являются **легковесные ФНФ типа арбитр** (Lightweight Secure PUF, LSPUF)(Рисунок 4) Легковесные ФНФ впервые были предложены в работе [15]. Они имеют более сложную структуру, чем XOR ФНФ. При такой конструкции ФНФ выдаёт -битовый ответ r, в котором каждый выходной бит генерируется путем сложения по модулю два ответов l экземпляров ФНФ типа арбитр. При этом в конструкции используются k $(k > l)$ экземпляров ФНФ. Биты ответа генерируются по следующей формуле: $r_i = \bigoplus_{j=0}^{l-1} y_{(i+s+j) \bmod k}$, здесь s – параметр сдвига, r_i – i-й бит общего ответа r, y_k – выходной бит k-ого экземпляра ФНФ-арбитр. Также в легковесных ФНФ на вход каждому экземпляру ФНФ подаются запросы, получаемые модификацией запроса x по некоторому алгоритму G. Для этой конструкции, как и для XOR ФНФ, существуют ограничения на число экземпляров ФНФ, появляющиеся из-за возрастания зашумлённости ответов при увеличении l.

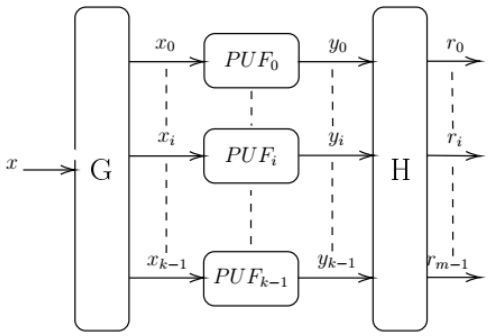

Рисунок 4 – конструкция легковесной ФНФ

Конструкция **с промежуточной ФНФ** (Interpose PUF, IPUF)(Рисунок 5) была предложена в работе [16]. Она строится следующим образом: сначала n-битный запрос $x = x_1 x_2 \dots x_n$ подаётся на вход k-XOR ФНФ, которая выдаёт один бит y_x. Затем этот ответ вставляется в середину запроса x и уже $(n+1)$ – битовый запрос $x' = x_1 x_2 \dots x_i y_x x_{i+1} \dots x_n$ подаётся на вход l-XOR ФНФ, которая возвращает ответ r. В результате теоретического анализа и экспериментальных результатов авторы заявили, что для конструкции $(k; l)$-промежуточной ФНФ характерны такие же аппаратные издержки и уровень надёжности, что и для $(k+l)$-XOR ФНФ, но при этом они обеспечивают гораздо более высокую устойчивость к моделирующим атакам. В частности, авторы демонстрируют, что (3; 3) – промежуточная ФНФ устойчива ко всем известным на тот момент атакам.

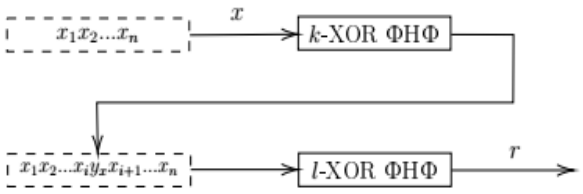

Рисунок 5. Конструкция с промежуточной ФНФ

IV. Генерация ключей на основе ФНФ

Требования безопасности диктуют, чтобы криптографические ключи, используемые в протоколах защиты информации, были достаточно случайными и непредсказуемыми. Поэтому для их создания необходимо использовать качественный источник случайности. Кроме того, на практике возникает проблема хранения криптографических ключей, так как именно на секретности ключа держится стойкость криптографических протоколов и механизмов. Лучше хранить ключи в какой-либо защищённой памяти, которая является дорогим ресурсом. ФНФ могут быть полезными в решении проблемы хранения и создания ключей. Случайность в ФНФ присутствует в самом устройстве и появляется из-за неконтролируемых производственных изменений, а использование защищенной энергонезависимой памяти оказывается необязательным, так как ответ ФНФ можно измерить заново каждый раз. Благодаря этим свойствам ФНФ находят своё применение в протоколах согласования/распределения и передачи секретных ключей [6,17–19].

Как уже говорилось в Разделе II, ответ экземпляра ФНФ puf_i на запрос x является случайной величиной $Y_i(x)$, заданной на множестве ответов Y_P. В дальнейшем в качестве множества ответов будет рассматриваться множество битовых векторов длины n. Конкретный ответ, полученный от экземпляра puf_i в ходе эксперимента, обозначается как y_i, и является битовой строкой длины n. Для устройств из класса ФНФ случайная величина $Y_i(x)$ может принимать различные значения, но с большой вероятность не сильно отличающиеся друг от друга в некоторой метрике $dist[;]$. Но чтобы воспроизводить ключ, необходимо каждый раз получать один и тот же ответ на фиксированный запрос, что на практике не выполняется. Поэтому возникает необходимость в методах, позволяющих получить исходный ответ $y_i \leftarrow puf_i(x)$ из близкого к нему ответа $y_i' \leftarrow puf_i(x)$. Также необходимо помнить, что ответы экземпляра ФНФ чаще всего принимают значения только из некоторого подмножества $\{0,1\}^n$, поэтому полученный таким образом ключ будет принимать не все возможные значения, что снижает уровень безопасности протоколов. Это требует использовать дополнительные механизмы для получения равномерно распределенного ключа.

Большинство протоколов, использующих ФНФ, для борьбы с возникновением ошибок на выходе ФНФ применяют нечёткие экстракторы. В [20] описана его концепция и предложена конкретная схема этого механизма.

A. Нечёткий экстрактор

Нечёткий экстрактор (fuzzy extractor) получает на вход ответ ФНФ и извлекает из него равномерно распределенную на $\boldsymbol{Y}$ строку k. Требуется, чтобы в случае, когда входные данные экстрактора отличаются от ранее полученного ответа ФНФ, но достаточно близки к нему, результат k его работы был тем же самым.

Определение 8. Разница между распределениями двух дискретных случайных величин A и B, которые принимают значения из одного и того же множества $\boldsymbol{Y}$, часто определяется количественно с помощью их *статистического расстояния*:

$$SD(A;B) \triangleq \frac{1}{2}\sum_{y\in\boldsymbol{Y}} |\Pr(A=y)-\Pr(B=y)|.$$

Определение 9. *Минимальная энтропия* $H_\infty(A)$ случайной величины A, заданной на множестве $\boldsymbol{Y}$**,** равна величине $-\log_2(\max_{y\in\boldsymbol{Y}}\Pr(A=y))$.

Определение 10. Нечёткий экстрактор $(\boldsymbol{Y}, m, l, t, \varepsilon)$ — это пара полиномиальных вероятностных алгоритмов, Gen и Rep. $Gen-$ алгоритм *генерации*, который принимает на вход $y \in \boldsymbol{Y}$ и возвращает битовую строку $k \in \{0,1\}^t$ и вспомогательную строку $h \in \{0,1\}^r$. Rep – алгоритм *восстановления*, который принимает на вход $y' \in \boldsymbol{Y}$, вспомогательную строку $h \in \{0,1\}^r$ и выдаёт результат $k \in \{0,1\}^t$. Нечёткий экстрактор должен удовлетворять следующим двум свойствам:

- *Свойство корректности* нечёткого экстрактора гарантирует, что если $dist[y; y'] \le t$, то $Rep(y', h) = k$.
- *Свойство безопасности* гарантирует, что для всех случайных величин A, заданных на $\boldsymbol{Y}$, с минимальной энтропией m выполняется: если $(K, H) \leftarrow Gen(A)$, то $SD\big((K,H);(U,H)\big) \le \varepsilon$, где U – равномерно распределенная на $\boldsymbol{Y}$ случайная величина.

Таким образом, нечёткий экстрактор является устойчивым к ошибкам, а также учитывает неоднородность данных, поэтому результат k будет иметь близкое к равномерному распределение, и его можно использовать в качестве ключа в криптографических приложениях.

Многие авторы протоколов не используют известные конструкции нечётких экстракторов, а под каждый класс ФНФ и под каждый протокол предлагают свои собственные [20,21].

B. Защищённый эскиз

Помимо нечёткого экстрактора используют ещё метод, называемый защищенным эскизом (secure sketch). Защищенный эскиз создает дополнительную информацию о выходном параметре y, которая не раскрывает информации об y, но в то же время позволяет точно восстановить y в случае незначительной ошибки при измерении. Таким образом, защищённый эскиз можно использовать для надежного воспроизведения подверженных шуму ответов ФНФ без риска для безопасности, присущего их хранению. В отличие от нечёткого экстрактора, защищённый эскиз не учитывает, что ответы ФНФ распределены не равномерно.

Определение 11. *Средняя условная минимальная энтропия* $\tilde{H}_\infty(A|B)$ случайной величины A относительно случайной величины B определяется следующим образом:

$\tilde{H}_\infty(A|B) \triangleq -\log_2[\sum_{b\in B} 2^{-H_\infty(A|B=b)}\Pr(B=b)]$.

Определение 12. Пусть $\boldsymbol{Y} = \{0,1\}^{\mathrm{n}}$, тогда $(\boldsymbol{Y}, m, m', \varepsilon)$-защищенный эскиз — это пара полиномиальных вероятностных алгоритмов SS и Rec. Алгоритм SS получает на вход битовую строку $y \in \boldsymbol{Y}$ и выдаёт битовую строку $h \in \{0,1\}^r$ $(h = SS(y))$, являющуюся вспомогательными данными. Алгоритм Rec принимает на вход элемент $y' \in \boldsymbol{Y}$, вспомогательные данные $h \in \{0,1\}^{\mathrm{r}}$ и возвращает ответ $y'' \in \boldsymbol{Y}$. Защищённый эскиз должен обладать следующими свойствами:

- *Свойство корректности* гарантирует, что если $dist[y, y'] \le \varepsilon$, то $Rec\,(y', SS(y)) = y$.
- *Свойство безопасности* гарантирует, что для всех случайных величин A, заданных на $\boldsymbol{Y}$, с минимальной энтропией m значение y может быть восстановлено противником, который знает h, с вероятностью не более, чем $2^{-m'}$, то есть $\tilde{H}_\infty(A|SS(A)) \ge m'$.

Необходимость использования нечётких экстракторов или защищённых эскизов существенно увеличивает время работы протокола и количество затрачиваемых ресурсов, а попытки их оптимизации часто приводят к возникновению уязвимостей [22]. Поэтому при проектировании протоколов эффективность и надёжность во многом зависит от методов, избавляющих от шума.

V. Протоколы аутентификации на основе ФНФ

На сегодняшний день предложено множество протоколов аутентификации, построенных на основе ФНФ, но при этом большинство из них имеет существенные недостатки, которые делают их применение на практике невозможным. Й. Дельво в работе 2017 года [23] проанализировал безопасность и эксплуатационные свойства 21 протокола аутентификации, основанных на ФНФ, и во всех выявил критические для эксплуатации недостатки. Автор выделяет только 9 протоколов (Таблица 1), которые являются перспективными для дальнейшего рассмотрения. При этом эти протоколы также имеют свои недостатки.

Обычно все протоколы аутентификации на основе ФНФ состоят из двух этапов: этапа регистрации, который выполняется в защищенной среде, и непосредственно самого этапа аутентификации, в течение которого взаимодействие уже выполняется по незащищенным каналам связи.

В этом разделе использованы следующие обозначения:

- $TRNG()$ – генератор истинно случайных чисел;
- $Hash()$ – бесключевая хэш-функция;
- $Hash_k()$ – ключевая хэш-функция;
- $HD(x, y)$ – расстояние Хэмминга.

В протоколах две стороны клиент A и сервер S.

Протоколы аутентификации, использующие ФНФ, можно разделить на два класса: протоколы парольной аутентификации и протоколы на основе запросов и ответов.

A. Протоколы парольной аутентификации

К первому типу относятся протоколы парольной аутентификации с генерацией ключей (паролей) на основе ФНФ. Построить такой протокол достаточно просто. Можно взять известный протокол аутентификации и использовать ФНФ для генерации случайного секрета (пароля). Хорошо проверенный протокол в сочетании с надёжным механизмом генерации ключей из ответов ФНФ, например с использованием нечёткого экстрактора, может обеспечить высокий уровень безопасности.

В качестве примера можно привести простейший протокол I-A аутентификации клиента, использующий криптографический алгоритм с ключом, в котором ключ хранится в защищенной памяти:

1. Этап регистрации.

   1.1 $A \leftarrow S : k_A$,

   Сервер генерирует случайный секретный ключ k_A и отправляет его клиенту по защищенному каналу.

   1.2 Клиент сохраняет этот ключ в защищенной памяти.

2. Аутентификация пользователя.

   2.1 $A \leftarrow S : n$,

   Сервер генерирует случайное значение n и отправляет его по незащищенному каналу;

   2.2 $A \rightarrow S : a = Hash_{k_A}(n)$

   2.3 Сервер сравнивает полученную величину с собственноручно вычисленным хэш-значением на ключе, хранящемся у него; если они совпадают, то аутентификация прошла успешно, иначе – ошибка.

Таблица 1 - Перспективные протоколы, отмеченные в работе [23]

Название протокола	Аутентификация клиента	Аутентификация сервера	Число аутентификаций
Протокол II-A	✓	×	∞
Контролируемый протокол [2,13,24,25]	✓	×	g (некоторое фиксированное число)
Протокол A-P. Садэги и др. [26]	✓	×	∞
Протокол с перевёрнутым нечётким экстрактором [5]	✓	✓	∞
Низкоресурсный протокол[27]	✓	×	∞
Протокол раздвоения шума [28]	✓	×	∞
Первый протокол блокировки [29]	✓	×	g
Второй протокол блокировки [29]	✓	✓	g

Можно построить протокол аутентификации аналогичный протоколу I-A, в котором ключ будет генерироваться из ответов ФНФ. Протокол II-A состоит из следующих этапов:

1. Этап регистрации.

   1.1 $A \rightarrow S: y = PUF()$,

   Клиент получает ответ экземпляра ФНФ, имеющегося у него, и отправляет ответ серверу по защищенному каналу;

   1.2 $S:(k, h) \leftarrow Gen(y)$

2. Аутентификация клиента.

   2.1 $A \leftarrow S: n, h$

   Сервер генерирует случайное значение n и посылает его вместе со вспомогательными данными h клиенту.

   2.2 $A \rightarrow S: a = Hash_{Rep(PUF(),h)}(n)$,

   Для аутентификации клиент вновь измеряет ответ ФНФ $y' = PUF()$. Затем с помощью вспомогательных данных, полученных от сервера, и алгоритма Rep клиент получает величину $k' = Rep(y', h)$, которую использует в качестве ключа хэш-функции. В итоге клиент получает хэш-значение a и посылает его серверу.

   2.3 Сервер выполняет проверку, если $a = Hash_k(n)$, то аутентификация прошла успешно, иначе – нет.

Главной проблемой протокола II-A является низкая эффективность, так как он использует нечёткий экстрактор и хэш-функцию, работа которых требует большого количества ресурсов. Также следует отметить, что для более высокого уровня безопасности необходимо проверять целостность вспомогательных данных, которые передаются по незащищённому каналу, что приведёт к ещё большим затратам.

ФНФ чаще всего используются в небольших устройствах, таких как RFID метки. В этих условиях предложенные протоколы оказываются непрактичными,

так как требуют большого количества ресурсов. Но их можно модифицировать разными способами для использования в конкретных задачах. Ниже будут приведены усовершенствованные протоколы парольной аутентификации на основе ФНФ.

a. Протокол Садэги

В некоторых задачах оказывается необходимым скрыть сам факт того, что клиент взаимодействует с сервером. В таких случаях сервер должен иметь возможность аутентифицировать клиента, но при этом не должен знать, кто именно с ним взаимодействует, то есть при этом сохраняется анонимность клиента. В работе [26] предложен протокол, решающий эту задачу. Протокол Садэги использует ключевые хэш-функции.

Протокол состоит из следующих этапов:

1. Этап регистрации.

   1.1 $A \leftarrow S: k_1$,

   Сервер генерирует случайный параметр k_1, который будет являться запросом для экземпляра ФНФ.

   1.2 $A \rightarrow S: y = PUF(k_1)$,

   Клиент сохраняет k_1 в незащищённой памяти, генерирует ответ ФНФ y на запрос k_1 и отправляет его серверу.

   1.3 $A \leftarrow S: h = SS(y)$

   Сервер вычисляет $k_2 = Hash(y)$. Затем он использует защищённый эскиз для генерации вспомогательных данных.

   1.4 Клиент сохраняет вспомогательные данные h в незащищенной памяти.

2. Аутентификация клиента.

   2.1 $A \leftarrow S : n_s$,

   Сервер генерирует случайное число n_s.

   2.2 $A \rightarrow S: n_A, Hash_{Hash(Rec(PUF(k_1),h))}(n_A, n_s)$,

   Клиент генерирует случайное число n_A. Затем подаёт на вход ФНФ запрос k_1. После, с помощью алгоритма Rec и вспомогательных данных извлекает $y' = Rec(PUF(k_1), h)$ и вычисляет его хэш-значение k_2'. Затем клиент хэш-значение от чисел n_A и n_s на ключе k_2'.

   2.3 Сервер последовательно сравнивает полученное значение с хэш-значениями, посчитанными им самим при использовании всех имеющихся у него ключей $k_{2,j}$, если ни одно не совпало, то – ошибка аутентификации.

В этом протоколе считается, что физические агрессивные (invasive) атаки, которые восстанавливают содержимое (k, h) из незащищенной памяти устройства, являются разрушительными, т. е. после них клиент больше не может участвовать в работе системы.

Недостатком протокола является то, что при большом количестве клиентов возникают большие задержки, так как серверу необходимо провести проверку полученного значения с хэш-значениями, полученными при использовании всех имеющихся у него ключей, соответствующих зарегистрированным клиентам.

b. Протокол с перевернутым нечётким экстрактором

В большинстве протоколов, предложенных до 2012 года, клиент выполнял алгоритм восстановления данных, а эта процедура является ресурсоёмкой. Для упрощения действий на стороне клиента в работе [30] предлагается использовать алгоритм восстановления на стороне сервера. Позже Р. Маэс, который был соавтором оригинального протокола, предложил в своей диссертации [5] модифицированную версию протокола, которая имеет лучше характеристики, чем исходный. Протокол Маэса предоставляет возможность двусторонней аутентификации следующим образом:

1. Этап регистрации.

   1.1 $A \leftarrow S: i_A$,

   Сервер присваивает клиенту идентификатор i_A. Затем он отправляет идентификатор клиенту по защищенному каналу.

   1.2 $A \rightarrow S: y = PUF()$,

   Клиент сохраняет в незащищенной памяти свой идентификатор, затем получает ответ ФНФ и отправляет его серверу.

2. Этап аутентификации.

   2.1 $A \rightarrow S: i, h = SS(PUF()), n_A$,

   Клиент измеряет ответ ФНФ $y' = PUF()$, вычисляет вспомогательные данные и посылает их серверу вместе со своим идентификатором и сгенерированным случайным числом n_A.

   2.2 $A \leftarrow S: Hash(i, h, Rec(y, h), n_A, n_s), n_s$,

   Сервер проверяет идентификатор. Если он корректен, то с помощью процедуры восстановления из ответа ФНФ , хранящегося у него вместе с идентификатором, и вспомогательных данных получает величину $y'' = Rec(y, h)$, генерирует случайное число n_s и вычисляет хэш-значение b от i, h, y'' , n_A и n_S.

   2.3 $A \rightarrow S: Hash(i, y', n_s)$

   Клиент проверяет хэш-значение, и в случае, когда оно верное, считает новое хэш-значение от идентификатора, ответа ФНФ и n_S.

   2.4 Сервер сверяет полученное хэш-значение со значением, посчитанным собственноручно, если всё верно, то аутентификация прошла успешно.

Проблема описанного протокола заключается в том, что он полагается на возможность многократного использования защищенного эскиза, но в [23] продемонстрирована нестойкость такого подхода.

B. Протоколы аутентификации на основе запросов и ответов

Ко второму типу относятся протоколы аутентификации на основе пар запрос-ответ ФНФ. В таких протоколах используются классы ФНФ, имеющие достаточно большое пространство пар запрос-ответ.

Концепцию таких протоколов можно проиллюстрировать на базовом протоколе аутентификации.

Подробнее этот протокол можно описать следующим образом:

1. Этап регистрации.

   (1.1– 1.g) Сбор пар запрос-ответ

   ___(i) $A \leftarrow S : c_i = TRNG()$

   Сервер генерирует случайный допустимый запрос c_i и отправляет его клиенту.

   __(ii) $A \rightarrow S: y_i = PUF(c_i)$

   Клиент подаёт число, полученное от сервера, на вход экземпляру ФНФ и получает ответ y_i, который посылается серверу.

   (1.g+1) Сервер присваивает переменной g_A значение g, определяющее количество возможных аутентификаций.

2. Аутентификация клиента (не более g раз).
   2.1 $A \leftarrow S : c_{g_A}$
   Если $g_A > 0$, сервер выбирает пару запрос-ответ под номером g_A и отправляет клиенту соответствующий запрос.
   2.2 $A \rightarrow S: y'_{g_A} = PUF(c_{g_A})$
   2.3 Сервер сравнивает полученный результат с ответом ФНФ, хранящемся у него, если $HD\left(y_{g_A}, y'_{g_A}\right) < \varepsilon$, то аутентификация прошла успешно, иначе ошибка. $g_A = g_A - 1$

Предложенный протокол не предполагает никакой защиты от атак, использующих технику машинного обучения (см Раздел VI). Злоумышленник, получив большое количество пар запрос-ответ, может построить математическую модель экземпляра ФНФ и выдать себя за клиента.

a. Контролируемый протокол на основе ФНФ

Чтобы противостоять атакам машинного обучения (VI), Б. Гасенд и др. [2,13,24,25] представили довольно общую концепцию контролируемых протоколов на основе ФНФ (Controlled PUF), которые по сути являются модификацией протокола базовой аутентификации. В этом разделе будет рассмотрен самый известный контролируемый протокол. Он состоит из следующих этапов:

1. Этап регистрации.

   (1.1– 1.g) Сбор пар запрос-ответ

   ___(i) $A \leftarrow S : c_i$

   Сервер генерирует случайный допустимый запрос c_i и отправляет его клиенту.

   __(ii) $A \rightarrow S: y_i = PUF(Hash(c_i))$

   Клиент вычисляет хэш-код от запроса, полученного от сервера, подаёт хэш-код на вход экземпляру ФНФ и отправляет серверу его ответ y_i.

   __(iii) Сервер использует алгоритм SS защищенного эскиза для получения вспомогательных данных $h_i = SS(y_i)$. Потом он вычисляет хэш-код $a_i = Hash(y_i, Hash(c_i))$, который сохраняется в памяти.

   (1.g+1) Сервер присваивает переменной g_A значение g, определяющее количество возможных аутентификаций.

2. Аутентификация клиента (не более g раз).
   2.1 $A \leftarrow S : c_{g_A}, h_{g_A}$
   2.2 $A \leftarrow S : Hash(Rec(PUF(c'), h_{g_A}), c')$
   Клиент подаёт на вход ФНФ хэш-значение от запроса $c' = Hash(c_{g_A})$, а затем с помощью вспомогательных данных и алгоритма Rec защищенного эскиза восстанавливает величину $y'' = Rec(y', h_{g_A})$. Затем вычисляет хэш-значение от восстановленного ответа y'' и хэш-значение запроса.
   2.3 Сервер сравнивает хэш-значение с величиной, хранящейся у него. Если они совпадают, то аутентификация прошла успешно, иначе ошибка.

В этом протоколе первая криптографическая хэш-функция, которая предшествует запросу к экземпляру ФНФ, не позволяет злоумышленнику произвольно выбирать запросы. Тем не менее протокол успешно противостоит атакам машинного обучения за счет второй хэш-функции, которая принимает на вход ответ сильной ФНФ и «маскирует» его.

Но для того чтобы можно было подавать на вход хэш-функции ответ ФНФ, требуется дополнительный модуль исправления ошибок. В противном случае даже одной ошибки в воспроизведенном отклике y'_i было бы достаточно, чтобы вызвать сбой аутентификации. Для устранения ошибок в ответах ФНФ используется защищенный эскиз, а сопутствующие вспомогательные данные хранятся на сервере.

Главным недостатком контролируемого протокола является то, что для его выполнения требуется большое количество ресурсов. Хотя контролируемые протоколы и уступают в эффективности протоколу II-A , они являются более стойкими, так как в них нет единого ключа (фиксированного количества бит), компрометация которого подрывает безопасность всей системы. В описанном протоколе для моделирования поведения экземпляра ФНФ злоумышленник должен собрать достаточно большое количество пар запрос-ответ.

В зависимости от конкретного класса ФНФ, используемого в протоколе, сложность успешных атак на этот протокол может существенно увеличиваться. Г. Беккер и Р. Кумар [31] построили атаку на этот тип протоколов, в котором использовалась ФНФ типа арбитр.

b. Низкоресурсный протокол на основе ФНФ

Аутентификация с помощью низкоресурсного протокола (slender PUF), предложенного М. Мажзуби и др. [27]. В этом протоколе клиент из ответа экземпляра ФНФ выбирает случайную подстроку и передаёт её серверу, тем самым он маскирует связь между запросами и ответами. В отличие от злоумышленника, сервер может проверить правильность полученных данных, так как на этапе регистрации с помощью машинного обучения была построена модель ФНФ.

1. Этап регистрации.

   1.1 На этапе регистрации сервер с помощью методов машинного обучения строит модель ФНФ.

2. Аутентификация клиента.

   2.1 $A \leftarrow S : c_s$

   Сервер генерирует случайное число c_S и отправляет его клиенту.

   2.2 $A \rightarrow S : c_A, SubCirc(PUF(c_s, c_A), n)$

   Клиент генерирует собственное случайное значение c_A и подаёт на вход ФНФ оба числа. Потом он снова генерирует случайное число n ($n \in [1, \lambda]$). Это значение определяет какие биты будут извлечены из ответа ФНФ. Алгоритм $SubCirc$ принимает на вход ответ ФНФ и n и выбирает η бит из ответа ФНФ длины λ бит следующим образом: $SubCirc\big((r_1 r_2 \dots r_\lambda), n\big) = \big(r_n\, r_{(n \bmod \lambda)+1} \dots r_{(n+\eta-2 \bmod \lambda)+1}\big), n \in [1, \lambda]$. Полученный вектор и своё случайное число c_A клиент отправляет серверу.

   2.3 Сервер, имея два случайных значения, подаёт их на вход модели ФНФ, а потом перебирает все возможные n. Если для любого n результат алгоритма $SubCirc(y', n)$ отличается от a больше, чем на некоторое ε, то выдаётся ошибка аутентификации.

Рассматриваемый протокол увеличивает сложность атак, использующих методы машинного обучения, но не исключает их возможность. В этом протоколе с одной стороны ФНФ должна быть легко моделируема нейронной сетью для построения у сервера модели ФНФ, но с другой стороны это может привести к уязвимости протокола по отношению к атакам, использующим технику машинного обучения. Поэтому в этой схеме следует использовать комбинацию ответов нескольких ФНФ, например XOR ответов ФНФ. Во время регистрации каждый экземпляр ФНФ может быть изучен сервером отдельно, и построена модель каждого экземпляра, а для злоумышленника это невозможно, так как по незащищённому каналу передаются только биты суммарного вектора. Однако нельзя быть абсолютно уверенным, что нет работающей атаки машинного обучения на этот протокол.

Ещё одной слабостью описанного протокола является чувствительность к генератору случайных чисел. К примеру генератор, который используется при проверке протокола, может позволить злоумышленнику выдать себя за клиента [23].

В 2014 году была предложена модификация низкоресурсного протокола на основе ФНФ, так называемый протокол раздвоения шума (noise bifurcation) [28].

Протокол раздвоение шума мало чем отличается от низкоресурсного протокола. В нём вместо алгоритма $SubCirc(y, n)$ для извлечения битов используется алгоритм $Decimate(y, n)$, который разделяет ответ y' на q частей размера λ и выбирает один бит из каждой части (из i-ой части он выбирает бит, соответствующий номеру n_i, где n_i выбираются независимо, случайно и равномерно из $[1, \lambda]$) $Decimate\big(x, n_1, n_2, \dots n_q\big) = \big(x_{n_1}\, x_{n_2+\lambda} \dots x_{n_q+(q-1)\lambda}\big)$.

Опять же, атакам машинного обучения противостоит то, что биты из ответов извлекаются случайным образом, и одному и тому же ответу будут соответствовать разные вектора a, пересылаемые по незащищенному каналу.

В отличие от злоумышленника, сервер может распутать связь между запросами и ответами при условии, что модель сильного ФНФ была построена на этапе регистрации.

Стойкость этого протокола определяется качество датчика случайных чисел. Что касается атак машинного обучения, то их применимость во многом зависит от конструкции ФНФ. Для этого протокола также, как и для легковесного протокола на основе ФНФ, удобнее всего использовать XOR ответов нескольких ФНФ, так как сервер на этапе построения модели может исследовать их по отдельности.

c. Протоколы блокировки на основе ФНФ

В работе [29] были предложены два протокола, которые получили название протоколов блокировки (Lockdown PUF), потому что клиент в ходе исполнения протокола проверяет подлинность сервера и блокирует его (не выполняет дальнейшие шаги аутентификации), если он не предъявляет правильную информацию.

Первый самый базовый протокол блокировки состоит из следующих этапов:

1. Этап регистрации.

   1.0 $A \leftarrow S : i_A$

   Сервер генерирует случайный идентификатор для клиента и отправляет его по защищенному каналу. Клиент сохраняет свой идентификатор в незащищенной памяти.

   (1.1 – 1.g) Сбор пар запрос-ответ

   ___(i) $A \leftarrow S : i$

   Сервер посылает клиенту числа от 1 до g, представленные в виде битовой строки.

   _(i+1) $A \rightarrow S : \big(y_{1,i}, y_{2,i}\big) = PUF(i)$

   Клиент подаёт на вход ФНФ запрос и получает ответ ФНФ. Ответ ФНФ клиент разделяет на две подстроки $y_{1,i}$ и $y_{2,i}$, которые отправляет серверу.

   1.g+1 Переменной g_A присваивается значение g.

2. Аутентификация клиента (не более g раз).

   2.1 $A \rightarrow S: i_A$

   Клиент отправляет свой идентификатор серверу.

   2.2 $A \leftarrow S: c, y_1$

   Сервер проверяет идентификатор, если идентификатор неправильный, то ошибка. После сервер получает битовую строку c из числа g_A и переменным (y_1, y_2) присваиваются значения $(y_{1,c}, y_{2,c})$ и уменьшается на 1 значение $g_A = g_A - 1$.

   2.3 $A \rightarrow S: y_2{}'$

   Клиент получает ответ ФНФ $(y_1', y_2') = PUF(c)$. Если $HD(y_1, y_1') > \varepsilon$, то происходит блокировка, сервер считается фальшивым. Если же векторы отличаются в небольшом количестве битов, то серверу отправляется вторая часть ответа ФНФ $y_2{}'$.

   2.4 Сервер сравнивает полученную часть, с имеющейся у него. Если $HD(y_2, y_2') < \varepsilon$, то аутентификация прошла успешно.

Реализация протокола блокировки на основе ФНФ с большим количеством пар запрос-ответ является модификацией протокола базовой аутентификации, которая избавлена от некоторых недостатков оригинала. Злоумышленник больше не может свободно запрашивать ответы клиента на неограниченное количество запросов.

Для этого протокола на практике серьезной проблемой являются атаки по побочным каналам в сочетании с методами машинного обучения. Несложно заметить, что злоумышленник может заставить клиента оценить ответ (y_1, y_2) на фиксированный вызов практически неограниченное количество раз.

Для увеличения трудоемкости атак протокол был модифицирован. Основное отличие от первого протокола состоит в том, что клиенты теперь генерируют новый случайный вызов c_A во время каждого запуска. Во втором протоколе меняется этап регистрации, с помощью методов машинного обучения сервер строит математическую модель ФНФ. На этапе аутентификации имеются следующие отличия:

2.1 $A \rightarrow S: i_A, c_A$

2.2 $A \leftarrow S: c, y_1$,где $(y_1, y_2) = PUF(c_A, c)$

В этом случае протокол будет обеспечивать двустороннюю аутентификацию. Так как запрос генерируется теперь клиентом, то злоумышленник не может заставить клиента проверять ответ (y_1, y_2) на один и тот же запрос c неограниченное число раз. Во втором протоколе сервер предсказывает ответ на запрос, частично сгенерированный клиентом, поэтому в этом случае клиент может аутентифицировать сервер. Фактически он проверяет наличие у него математической модели экземпляра ФНФ.

VI. Моделирование ФНФ на основе технологии машинного обучения

Для того, чтобы экземпляры класса ФНФ могли успешно использоваться в криптографических приложениях, необходимо, чтобы они обладали свойством математической неклонируемости. Класс обладает этим свойством, если другие устройства не могут смоделировать поведение экземпляра класса.

Часто классы ФНФ можно промоделировать с помощью методов машинного обучения, тем самым нарушается свойство математической неклонируемости класса. В текущий момент времени разработчики активно предлагают различные конструкции, комбинирующие ответы ФНФ и способные противостоять машинному обучению, но они часто также оказываются нестойкими. В этом разделе будут перечислены некоторые конструкции ФНФ, успешно моделируемые с помощью машинного обучения.

Экземпляры ФНФ применяются для внутренней случайности, и считается, что никто не может предсказать ответ экземпляра, не имея доступ к нему. Но на практике это не так. Обычно ответы ФНФ достаточно сильно коррелируют с запросами, и, имея некоторое количество пар запрос-ответ, злоумышленник может построить математическую модель ФНФ, используя методы машинного обучения. Поэтому при построении криптографической системы нужно следить за тем, чтобы злоумышленники не имели доступ к большому количеству пар запрос-ответ.

Конструкция ФНФ типа арбитр была описана в Разделе III, в силу своей простоты реализации она является привлекательной для использования на практике. В ФНФ типа арбитр задержка цифрового канала моделируется как сумма задержек его компонентов. Тогда каждая пара запрос-ответ представляет собой линейное неравенство с неизвестными параметрами задержки и может использоваться для получения более точного приближения. То есть по факту разность задержек определяется следующим образом: $\delta_x = \vec{w}^T \Phi$. Где $\vec{w}$ – $(n+1)$-мерный вектор весов, состоящий из действительных чисел. Его компоненты зависят от задержки элементов пути. Ф – это вектор чётности, формирующийся на основе запроса x следующим образом: $\Phi[n] = 1$, $\Phi[i] = \prod_{j=i}^{n-1}(1 - 2x_j)$, $i = 0, \dots, n-1$. Успешное моделирование ФНФ типа арбитр сводится к точному предсказанию вектора веса $\vec{w}$. Это может быть достигнуто довольно успешно с помощью нескольких алгоритмов машинного обучения [10–12].

На практике в качестве ФНФ с большим количеством запрос-ответ чаще всего используются классы, комбинирующие ответы нескольких экземпляров ФНФ типа арбитр, такие как XOR-ФНФ, промежуточная-ФНФ и другие. Такие классы являются наиболее исследованными с точки зрения математического моделирования. В Таблице 2 приведен анализ применимости методов машинного обучения для моделирования некоторых классов ФНФ из работ [14,32,33].

Нарушение свойства математической неклонируемости ФНФ приводит к атакам на протоколы аутентификации, использующие эти ФНФ. В работе [34]

приведены атаки на 5 протоколов аутентификации [35–39], использующие ФНФ типа арбитр. В этих протоколах были недостаточно скрыты ответы ФНФ. Поэтому при построении криптографической системы, использующей ФНФ, необходимо хранить таблицу пар запрос-ответ в секрете и учитывать количество информации, которое можно передавать по открытому каналу, а также анализировать алгоритм на стойкость по отношению к известным атакам машинного обучения.

Таблица 2 – Атаки на классы, построенные на основе ФНФ типа арбитр

Тип конструкции	Метод машинного обучения	Размер запроса	Число пар запрос-ответ, на которых обучалась модель	Успешность атак (%)	Время обучения	Реализация ФНФ
Легковесная ФНФ-арбитр(5-XOR)	Глубокое машинное обучение	128	1 200 000	96.22	3 часа 11 мин	симуляция
(4,4)-промежуточная ФНФ-арбитр	Глубокое машинное обучениие	128	647 000	97.68	32 мин 17 с	симуляция
5-XOR ФНФ-арбитр	Глубокое машинное обучение	128	655 000	97.87	29 мин 21 с	симуляция
Легковесная ФНФ-арбитр (6-XOR)	Глубокое машинное обучение	64	800 000	97.42	33 мин 24 с	симуляция
6-XOR ФНФ-арбитр	Глубокое машинное обучение	64	680 000	97.68	20 мин 52 с	симуляция
(4,4)-промежуточная ФНФ-арбитр	Глубокое машинное обучение	64	320 000	97.44	5 мин 23 с	симуляция
ФНФ-арбитр	Логистическая регрессия	128	39 200	99.9	2.3 с	симуляция
		64	6 500	99	0.83 с	ПЛИС
		64	6 500	99	0.76 с	ИС
5-XOR ФНФ-арбитр	Логстическая регрессия	128	500 000	99	16 ч 36 мин	симуляция
		64	78 000	99	39 мин	ПЛИС
		64	78 000	99	18 ч 9 мин	ИС
Легковесная ФНФ (5-XOR)	Логистическая регрессия	128	1 000 000	99	267 дней	симуляция

VII. Заключение

В работе рассмотрены базовые подходы к построению ФНФ, описаны основные протоколы аутентификации, которые в своей конструкции используют ФНФ.

Аанализ протоколов аутентификации показывает, что на сегодняшний день они не обеспечивают необходимый уровень безопасности. Большинство классов ФНФ оказываются уязвимыми к атакам на основе методов машинного обучения, а меры, противостоящие этим атакам, значительно усложняют протоколы. Поэтому преждевременно рассматривать ФНФ в качестве структурного узла перспективных легковесных криптографических механизмов и протоколов. Рекомендуется использовать ФНФ только в задачах аутентификации устройств, в которых модель нарушителя не позволяет реализовывать атаки машинного обучения, например, в устройствах, которые достаточно редко работают в активном режиме и к ним не требуется частое обращение.

VIII. Библиография

1. Pappu R. Physical One-Way Functions // Science. 2002. Vol. 297, № 5589. P. 2026–2030.
2. Gassend B. et al. Silicon physical random functions // Proceedings of the 9th ACM conference on Computer and communications security - CCS ’02. Washington, DC, USA: ACM Press, 2002. P. 148.
3. Security with Noisy Data / ed. Tuyls P., Skoric B., Kevenaar T. London: Springer London, 2007.
4. Zynq UltraScale+ MPSoC Device: Technical Reference Manual.
5. Maes R. Physically Unclonable Functions:Constructions, Properties and Applications. 2012. P. 260.

6. Guajardo J. et al. FPGA Intrinsic PUFs and Their Use for IP Protection // Cryptographic Hardware and Embedded Systems - CHES 2007 / ed. Paillier P., Verbauwhede I. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. Vol. 4727. P. 63–80.
7. Rührmair U., Busch H., Katzenbeisser S. Strong PUFs: Models, Constructions, and Security Proofs // Towards Hardware-Intrinsic Security / ed. Sadeghi A.-R., Naccache D. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. P. 79–96.
8. Daihyun Lim et al. Extracting secret keys from integrated circuits // IEEE Trans. VLSI Syst. 2005. Vol. 13, № 10. P. 1200–1205.
9. Lee J.W. et al. A technique to build a secret key in integrated circuits for identification and authentication applications // 2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525). Honolulu, HI, USA: Widerkehr and Associates, 2004. P. 176–179.
10. Ganji F., Tajik S., Seifert J.-P. PAC learning of arbiter PUFs // J Cryptogr Eng. 2016. Vol. 6, № 3. P. 249–258.
11. Nguyen P.H. et al. Security Analysis of Arbiter PUF and Its Lightweight Compositions Under Predictability Test // ACM Trans. Des. Autom. Electron. Syst. 2016. Vol. 22, № 2. P. 1–28.
12. R ührmair U. et al. Modeling attacks on physical unclonable functions // Proceedings of the 17th ACM conference on Computer and communications security - CCS '10. Chicago, Illinois, USA: ACM Press, 2010. P. 237.
13. Gassend B.L.P. Physical Random Functions. P. 89.
14. Santikellur P., Bhattacharyay A., Chakraborty R.S. Deep Learning based Model Building Attacks on Arbiter PUF Compositions. P. 10.
15. Majzoobi M., Koushanfar F., Potkonjak M. Lightweight secure PUFs // 2008 IEEE/ACM International Conference on Computer-Aided Design. San Jose, CA, USA: IEEE, 2008. P. 670–673.
16. Nguyen P.H. et al. The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks. P. 48.
17. Usmani M.A. et al. Efficient PUF-Based Key Generation in FPGAs Using Per-Device Configuration // IEEE Trans. VLSI Syst. 2019. Vol. 27, № 2. P. 364–375.
18. Yu M.-D. et al. Lightweight and Secure PUF Key Storage Using Limits of Machine Learning // Cryptographic Hardware and Embedded Systems – CHES 2011 / ed. Preneel B., Takagi T. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. Vol. 6917. P. 358–373.
19. Zhang J., Qu G. Physical Unclonable Function-based Key Sharing via Machine Leaning for IoT Security // IEEE Trans. Ind. Electron. 2019. P. 1–1.
20. Dodis Y., Reyzin L., Smith A. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. P. 18.
21. Kang H. et al. Cryptographie key generation from PUF data using efficient fuzzy extractors // 16th International Conference on Advanced Communication Technology. Pyeongchang, Korea (South): Global IT Research Institute (GIRI), 2014. P. 23–26.
22. Merli D. et al. Side-Channel Analysis of PUFs and Fuzzy Extractors // Trust and Trustworthy Computing / ed. McCune J.M. et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. Vol. 6740. P. 33–47.
23. Delvaux J. Security Analysis of PUF-Based Key Generation and Entity Authentication. P. 258.
24. Gassend B. et al. Controlled physical random functions // 18th Annual Computer Security Applications Conference, 2002. Proceedings. Las Vegas, NV, USA: IEEE Comput. Soc, 2002. P. 149–160.
25. Gassend B. et al. Controlled physical random functions and applications // ACM Trans. Inf. Syst. Secur. 2008. Vol. 10, № 4. P. 1–22.
26. Sadeghi A.-R., Visconti I., Wachsmann C. Enhancing RFID Security and Privacy by Physically Unclonable Functions // Towards Hardware-Intrinsic Security / ed. Sadeghi A.-R., Naccache D. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. P. 281–305.
27. Majzoobi M. et al. Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching // 2012 IEEE Symposium on Security and Privacy Workshops. San Francisco, CA, USA: IEEE, 2012. P. 33–44.
28. Yu M.-D. et al. A noise bifurcation architecture for linear additive physical functions // 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST). Arlington, VA, USA: IEEE, 2014. P. 124–129.
29. Yu M.-D. et al. A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication // IEEE Trans. Multi-Scale Comp. Syst. 2016. Vol. 2, № 3. P. 146–159.
30. Van Herrewege A. et al. Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-Enabled RFIDs // Financial Cryptography and Data Security / ed. Keromytis A.D. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Vol. 7397. P. 374–389.
31. Becker G.T., Kumar R. Active and Passive Side-Channel Attacks on Delay Based PUF Designs. P. 14.
32. Ruhrmair U. et al. PUF Modeling Attacks on Simulated and Silicon Data // IEEE Trans.Inform.Forensic Secur. 2013. Vol. 8, № 11. P. 1876–1891.
33. Ruhrmair U., Solter J. PUF modeling attacks: An introduction and overview // Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014. Dresden, Germany: IEEE Conference Publications, 2014. P. 1–6.
34. Delvaux J. Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF–FSMs // IEEE Trans.Inform.Forensic Secur. 2019. Vol. 14, № 8. P. 2043–2058.
35. Gao Y. et al. Obfuscated challenge-response: A secure lightweight authentication mechanism for PUF-based pervasive devices // 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops). Sydney, Australia: IEEE, 2016. P. 1–6.
36. Gao Y. et al. PUF-FSM: A Controlled Strong PUF // IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2017. P. 1–1.
37. Idriss T., Bayoumi M. Lightweight highly secure PUF protocol for mutual authentication and secret message

exchange // 2017 IEEE International Conference on RFID Technology & Application (RFID-TA). Warsaw: IEEE, 2017. P. 214–219.

38. Konigsmark S.T.C., Chen D., Wong M.D.F. PolyPUF: Physically Secure Self-Divergence // IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2016. Vol. 35, № 7. P. 1053–1066.
39. Ye J., Hu Y., Li X. RPUF: Physical Unclonable Function with Randomized Challenge to resist modeling attack // 2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST). Yilan, Taiwan: IEEE, 2016. P. 1–6.

Physically Unclonable Functions in cryptography

V. Belsky, I. Chizhov, A. Chichaeva, V. Shishkin

Abstract— **A physically unclonable function is a hardware device whose instances have several unique parameters and characteristics, i.e. it is impossible to create two instances with identical values of these characteristics due to the properties of the physical production process. We can assume that these characteristics take on random values. This kind of physical randomness can be used in various cryptographic protocols and mechanisms. Physically unclonable functions require few resources, so they are promising for use in devices with limited resources such as RFID tags.**

The paper investigates the possibility of using physically unclonable functions in cryptographic protocols to solve the following problems: generating random values, identification and authentication.

The advantage of using physically unclonable functions to generate random parameters is that the obtained values do not need to be stored in memory, because parameters can be regenerated on the fly. This is an excellent advantage since protocol parameters often need to be stored in secure memory, which is an expensive resource. However, some measurement errors often occur while obtaining physical devices' characteristics, so it is necessary to use some form of error-correction. The paper describes the basic constructions used for these purposes.

Today many authentication protocols based on physically unclonable functions have been proposed. They can be divided into two classes: password authentication protocols with key generation based on physically unclonable functions and authentication protocols based on challenge-response pairs. The article discusses the existing authentication protocols, their advantages and disadvantages.

The paper also considers the possibility of creating a mathematical model of physically unclonable functions. Modern machine learning methods allow us to create a mathematical "clone" of a device instance. This fact is a significant disadvantage of physically unclonable functions.

As a result, we conclude that physically unclonable functions are promising for use in devices with limited resources. At the same time, most of the currently proposed designs have a few practical disadvantages and are vulnerable to attacks based on machine learning methods. This fact suggests that it is too early to consider physically unclonable functions as a structural element of cryptographic mechanisms and protocols.

Keywords — **Physically unclonable functions, Authentication.**

REFERENCES

1. Pappu R. Physical One-Way Functions // Science. 2002. Vol. 297, № 5589. P. 2026–2030.
2. Gassend B. et al. Silicon physical random functions // Proceedings of the 9th ACM conference on Computer and communications security - CCS '02. Washington, DC, USA: ACM Press, 2002. P. 148.
3. Security with Noisy Data / ed. Tuyls P., Skoric B., Kevenaar T. London: Springer London, 2007.
4. Zynq UltraScale+ MPSoC Device: Technical Reference Manual.
5. Maes R. Physically Unclonable Functions:Constructions, Properties and Applications. 2012. P. 260.
6. Guajardo J. et al. FPGA Intrinsic PUFs and Their Use for IP Protection // Cryptographic Hardware and Embedded Systems - CHES 2007 / ed. Paillier P., Verbauwhede I. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. Vol. 4727. P. 63–80.
7. Rührmair U., Busch H., Katzenbeisser S. Strong PUFs: Models, Constructions, and Security Proofs // Towards Hardware-Intrinsic Security / ed. Sadeghi A.-R., Naccache D. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. P. 79–96.
8. Daihyun Lim et al. Extracting secret keys from integrated circuits // IEEE Trans. VLSI Syst. 2005. Vol. 13, № 10. P. 1200–1205.
9. Lee J.W. et al. A technique to build a secret key in integrated circuits for identification and authentication applications // 2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525). Honolulu, HI, USA: Widerkehr and Associates, 2004. P. 176–179.
10. Ganji F., Tajik S., Seifert J.-P. PAC learning of arbiter PUFs // J Cryptogr Eng. 2016. Vol. 6, № 3. P. 249–258.
11. Nguyen P.H. et al. Security Analysis of Arbiter PUF and Its Lightweight Compositions Under Predictability Test // ACM Trans. Des. Autom. Electron. Syst. 2016. Vol. 22, № 2. P. 1–28.
12. R ührmair U. et al. Modeling attacks on physical unclonable functions // Proceedings of the 17th ACM conference on Computer and communications security - CCS '10. Chicago, Illinois, USA: ACM Press, 2010. P. 237.
13. Gassend B.L.P. Physical Random Functions. P. 89.

14. Santikellur P., Bhattacharyay A., Chakraborty R.S. Deep Learning based Model Building Attacks on Arbiter PUF Compositions. P. 10.
15. Majzoobi M., Koushanfar F., Potkonjak M. Lightweight secure PUFs // 2008 IEEE/ACM International Conference on Computer-Aided Design. San Jose, CA, USA: IEEE, 2008. P. 670–673.
16. Nguyen P.H. et al. The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks. P. 48.
17. Usmani M.A. et al. Efficient PUF-Based Key Generation in FPGAs Using Per-Device Configuration // IEEE Trans. VLSI Syst. 2019. Vol. 27, № 2. P. 364–375.
18. Yu M.-D. et al. Lightweight and Secure PUF Key Storage Using Limits of Machine Learning // Cryptographic Hardware and Embedded Systems – CHES 2011 / ed. Preneel B., Takagi T. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. Vol. 6917. P. 358–373.
19. Zhang J., Qu G. Physical Unclonable Function-based Key Sharing via Machine Leaning for IoT Security // IEEE Trans. Ind. Electron. 2019. P. 1–1.
20. Dodis Y., Reyzin L., Smith A. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. P. 18.
21. Kang H. et al. Cryptographie key generation from PUF data using efficient fuzzy extractors // 16th International Conference on Advanced Communication Technology. Pyeongchang, Korea (South): Global IT Research Institute (GIRI), 2014. P. 23–26.
22. Merli D. et al. Side-Channel Analysis of PUFs and Fuzzy Extractors // Trust and Trustworthy Computing / ed. McCune J.M. et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. Vol. 6740. P. 33–47.
23. Delvaux J. Security Analysis of PUF-Based Key Generation and Entity Authentication. P. 258.
24. Gassend B. et al. Controlled physical random functions // 18th Annual Computer Security Applications Conference, 2002. Proceedings. Las Vegas, NV, USA: IEEE Comput. Soc, 2002. P. 149–160.
25. Gassend B. et al. Controlled physical random functions and applications // ACM Trans. Inf. Syst. Secur. 2008. Vol. 10, № 4. P. 1–22.
26. Sadeghi A.-R., Visconti I., Wachsmann C. Enhancing RFID Security and Privacy by Physically Unclonable Functions // Towards Hardware-Intrinsic Security / ed. Sadeghi A.-R., Naccache D. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. P. 281–305.
27. Majzoobi M. et al. Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching // 2012 IEEE Symposium on Security and Privacy Workshops. San Francisco, CA, USA: IEEE, 2012. P. 33–44.
28. Yu M.-D. et al. A noise bifurcation architecture for linear additive physical functions // 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST). Arlington, VA, USA: IEEE, 2014. P. 124–129.
29. Yu M.-D. et al. A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication // IEEE Trans. Multi-Scale Comp. Syst. 2016. Vol. 2, № 3. P. 146–159.
30. Van Herrewege A. et al. Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-Enabled RFIDs // Financial Cryptography and Data Security / ed. Keromytis A.D. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Vol. 7397. P. 374–389.
31. Becker G.T., Kumar R. Active and Passive Side-Channel Attacks on Delay Based PUF Designs. P. 14.
32. Ruhrmair U. et al. PUF Modeling Attacks on Simulated and Silicon Data // IEEE Trans.Inform.Forensic Secur. 2013. Vol. 8, № 11. P. 1876–1891.
33. Ruhrmair U., Solter J. PUF modeling attacks: An introduction and overview // Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014. Dresden, Germany: IEEE Conference Publications, 2014. P. 1–6.
34. Delvaux J. Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF–FSMs // IEEE Trans.Inform.Forensic Secur. 2019. Vol. 14, № 8. P. 2043–2058.
35. Gao Y. et al. Obfuscated challenge-response: A secure lightweight authentication mechanism for PUF-based pervasive devices // 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops). Sydney, Australia: IEEE, 2016. P. 1–6.
36. Gao Y. et al. PUF-FSM: A Controlled Strong PUF // IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2017. P. 1–1.
37. Idriss T., Bayoumi M. Lightweight highly secure PUF protocol for mutual authentication and secret message exchange // 2017 IEEE International Conference on RFID Technology & Application (RFID-TA). Warsaw: IEEE, 2017. P. 214–219.
38. Konigsmark S.T.C., Chen D., Wong M.D.F. PolyPUF: Physically Secure Self-Divergence // IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2016. Vol. 35, № 7. P. 1053–1066.
39. Ye J., Hu Y., Li X. RPUF: Physical Unclonable Function with Randomized Challenge to resist modeling attack // 2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST). Yilan, Taiwan: IEEE, 2016. P. 1–6.