UNIERSYTET WARSZAWSKI
WYDZIAŁ MATEMATYKI, INFORMATYKI I MECHANIKI

Michał Zając
Instytut Informatyki

EFFECTIVE CRYPTOGRAPHIC PROTOCOLS WITH LIMITED COMPUTATIONAL POWER AND MEMORY

autoreferat

promotor:
dr hab. Stefan Dziembowski
Instytut Informatyki

1 Wprowadzenie

1.1 Kryptografia odporna na wycieki – motywacje

U podstaw bezpieczeństwa większości współczesnych protokołów kryptograficznych leżą dwa założenia dotyczące wykorzystywanego klucza kryptograficznego. Po pierwsze jest on tajny[1], po drugie jest on losowy, tj. klucz jest próbką z rozkładu jednostajnie losowego.

Rozpatrzmy dwa najpopularniejsze schematy szyfrowania i podpisu cyfrowego kryptografii klucza publicznego, tj. RSA [36] i El Gamal [17]. Dowiedziono, że są one bezpieczne przy założeniu trudności tzw. problemu RSA (RSA) i trudności obliczenia logarytmu dyskretnego (El Gamal). Oba problemy są powszechnie uznawane za trudne, a złamanie jednego ze schematów wiązałoby się z rozwiązaniem któregoś z nich. Jednakże ani RSA, ani El Gamal nie są bezpieczne kiedy wykorzystywany w nich *klucz prywatny* nie jest tajny lub nie jest losowy. W takim przypadku, protokoły te, używane do szyfrowania, nie gwarantują poufności przesyłanych danych, a jako podpisy cyfrowe nie można na nich polegać chcąc rozstrzygać kwestię autentyczności dokumentów. Poniżej przedstawiono argumenty wskazujące, że założenie idealnie losowego, tajnego klucza nie zawsze jest spełnione.

Implementacja. Istotnym problemem związanym z rozpatrywaniem bezpieczeństwa protokołów kryptograficznych jest proces ich implementacji. Niskopoziomowe języki programowania, takie jak C, pozwalają na głęboką ingerencję w sposób działania komputera podczas wykonywania kodu programu. Z jednej strony pozwala to na efektywne wykonanie niezbędnych operacji i obliczeń. Z drugiej zaś nie można nie wspomnieć o niebezpieczeństwach jakie się z nimi wiążą. Każda pomyłka programisty może skutkować powstaniem luki bezpieczeństwa. Przykładowym atakiem jest tzw. *buffer overflow*. Za jego pomocą przeprowadzono w ostatnim czasie atak *Hearbleed*, który miał na celu pozyskanie haseł z serwerów obsługujących sesje TLS/SSL [7, 37]. Niestety, wykorzystanie języków wysokopoziomowych nie gwarantuje tego, że kod będzie bezpieczny [6, 25, 30].

Wirusy i złośliwe oprogramowanie. Oprócz zagrożeń wynikających z błędów po stronie programistów, musimy zwrócić uwagę na fakt, że protokoły kryptograficzne działają w pewnym środowisku tj. w systemie operacyjnym. Ponadto urządzenie, na którym wykonywane są obliczenia, posiada dostęp do internetu. Dostęp do internetu naraża urządzenie na wszelkiego rodzaju wirusy, konie trojańskie, czy ataki hakerów, które mogą pozyskać częściowe informacje na temat klucza.

Zagrożenia związane z możliwością pozyskania tajnego klucza za pomocą złośliwego oprogramowania można uznać za szczególnie istotne dla współczesnej kryptografii. W rozprawie skupiamy się w szczególności na przeciwdziałaniu ww. zagrożeniom. Zagrożenia te w szczególny sposób wpłynęły na wybór modelu, w którym opracowane zostały wyniki rozprawy. Poniżej przedstawiamy przykładowe ataki, które mogą zostać przeprowadzone przez wirus komputerowy.

W dalszej części autoreferatu używamy litery $\mathcal{A}$ do oznaczenia adwersarza, tj. maszyny Turinga, której zadaniem jest złamanie bezpieczeństwa opisywanego przez nas schematu kryptograficznego. W poniższych przykładach, $\mathcal{A}_{\text{creator}}, \mathcal{A}_{\text{virus}}$ to dwie probabilistyczne maszyny działające w czasie wielomianowym. Obydwie posiadają dostęp do dodatkowej taśmy `aux` zawierającej losowe bity. Klasę takich maszyn oznaczamy przez PPT (ang. *probabilistic polynomial time*).

[1] W przypadku kryptografii klucza publicznego mowa tu o tajności klucza prywatnego.

Przykład 1 (Kryptografia a złośliwe oprogramowanie)**.** *Załóżmy, że $\mathcal{A}_{virus}$, znajduje się na maszynie $\mathcal{C}$ i ma dostęp do pamięci urządzenia. W szczególności, ma dostęp do miejsc, w których składowany jest klucz kryptograficzny K używany przez autoryzowanego użytkownika komputera do identyfikowania się w banku. Załóżmy następujący scenariusz działań wirusa $\mathcal{A}_{virus}$:*

1. *znajduje w pamięci $\mathcal{C}$ klucz K;*
2. *łączy się z bankiem za pomocą klucza K;*
3. *żąda transferu wszystkich zgromadzonych na rachunku środków na konto zarządzane przez $\mathcal{A}_{creator}$.*

Przykład 2 (Kryptografia a złośliwe oprogramowanie (2))**.** *Załóżmy, że $\mathcal{A}_{virus}$, znajduje się na maszynie $\mathcal{C}$ i ma dostęp do pamięci urządzenia. W szczególności, ma dostęp do miejsc, w których składowany jest klucz kryptograficzny K używany przez autoryzowanego użytkownika komputera do identyfikacji w banku. Załóżmy następujący scenariusz działań wirusa $\mathcal{A}_{virus}$:*

1. *znajduje w pamięci $\mathcal{C}$ prywatny klucz K;*
2. *publikuje klucz K na ogólnodostępnej tablicy internetowej (*bulletin board*), gdzie może zostać znaleziony przez $\mathcal{A}_{creator}$;*

Zauważmy, że w przypadku, kiedy na maszynie rezyduje wirus mający dostęp do tajnych kluczy, praktycznie żadne zabezpieczenia kryptograficzne (nie wykorzystujące np. dodatkowych zewnętrznych urządzeń) nie są możliwe. Wynika to z prostego powodu — jakiegokolwiek schematu kryptograficznego byśmy nie stosowali, jakkolwiek mocny klucz byśmy stworzyli, to wirus i tak może mieć dostęp do tej części pamięci, w której jest on przechowywany i może użyć go w dowolnym celu. Z drugiej strony, możemy założyć, że maszyna w pewnych momentach jest wolna od wirusa, który np. zostaje usunięty za pomocą oprogramowania antywirusowego a potencjalny haker, wyloguje się z niej.

Drugi przykład pokazuje, że nawet wtedy, kiedy złośliwe oprogramowanie zostanie usunięte z dysku komputera, może okazać się, że spowodowało ono nieodwracalne szkody. Twórca wirusa nie potrzebuje go więcej. Uzyskał informację jakiej potrzebował – klucz prywatny użytkownika komputera. Teraz może zidentyfikować się w banku nawet wtedy, kiedy nie ma dostępu do maszyny.

Zebrane w rozprawie wyniki zakładają następujący scenariusz – co prawda nie jesteśmy w stanie zapewnić bezpieczeństwa kiedy kontrolę nad maszyną sprawuje wirus lub haker, jednak możemy zapewnić bezpieczeństwo, kiedy wirus zostanie usunięty a haker wyloguje się. W szczególności pokazujemy, że złośliwe oprogramowanie nie może przekazać swojemu twórcy informacji niezbędnych do wykonywania czynności wymagających klucza prywatnego w imieniu autoryzowanego użytkownika. Szczegóły na temat modelu obliczeń przedstawiono poniżej.

1.2 Bounded Retrieval Model

Wyniki rozprawy przygotowano w modelu *Bounded Retrieval Model* (BRM) zaproponowanym niezależnie przez Dziembowskiego [13] i Di Crescenzo [8] a następnie rozwijanym np. w [13, 1, 2, 10, 16, 14, 33, 15]. BRM jest szczególnie użyteczny przy projektowaniu protokołów kryptograficznych, mających zapewniać bezpieczeństwo nawet wtedy, kiedy komputer zostanie zaatakowany przez wirus, czy inne złośliwe oprogramowanie.

Parametr wycieku. W modelu tym przyjmuje się, że adwersarz atakujący system nie jest w stanie pozyskać więcej informacji o tajnym kluczu niż przyjęty *parametr wycieku* λ. Dokładniej, zakładamy że przeciwnik nie może pozyskać o kluczu więcej niż λ bitów informacji. W modelu BRM wymagamy by protokół kryptograficzny pozostawał bezpieczny nawet w takim przypadku.

Odpowiedni dobór parametru wycieku ma kluczowe znaczenie. Kiedy jest on bardzo mały, nie możemy twierdzić, że dobrze oddaje on rzeczywistość. Rezydujący na urządzeniu wirus może bez większych przeszkód wysłać wiele megabajtów czy nawet gigabajty danych bez wzbudzania większych podejrzeń. Zauważmy, że kiedy parametr wycieku sięga kilku gigabajtów, przechowywany na urządzeniu klucz kryptograficzny musi być znacząco większy. Wynika to z prostej obserwacji. Jeżeli długość klucza jest mniejsza niż λ, przeciwnik może *wyciec* go w całości.

Niestety, konieczność stosowania kluczy prywatnych długości gigabajtów sprawia, że nie jest możliwe zastosowanie powszechnie znanych schematów kryptograficznych jak choćby RSA czy El Gamal. Schematy te, dla kluczy podanej wielkości, są po prostu zbyt niewydajne by mogły zostać zastosowane w praktyce.

Bardziej formalnie można powiedzieć, że przeciwnik $\mathcal{A}$ wyposażony w wyciek co najwyżej λ działa następująco. Oznaczmy, że $\mathtt{sk}$ oznacza tajny klucz, o którym $\mathcal{A}$ chce dostać informacje. $\mathcal{A}$ jest wyposażony w dostęp do wyroczni losowej $\mathcal{O}(\mathtt{sk})$. Zapytania jakie $\mathcal{A}$ może wysłać to funkcje $f_i : \{0,1\}^{|\mathtt{sk}|} \to \{0,1\}^{\lambda_i}$. W szczególności $\mathcal{A}$ może zdefiniować funkcję f_j po poznaniu wartości $f_i(\mathtt{sk})$ dla $i < j$. Wyrocznia wyciekowa $\mathcal{O}(\mathtt{sk})$ na zapytanie f_j zwraca wartość $f_j(\mathtt{sk})$ wtedy i tylko wtedy, kiedy $\sum_{i \leqslant j} \lambda_i \leqslant \lambda$. Zauważmy, że przeciwnik $\mathcal{A}$ nie jest ograniczony do poznawania kolejnych bitów klucza prywatnego. Może on poznać wartość dowolnej funkcji z $\mathtt{sk}$ jako argumentem.

Z kluczami w BRM wiążą się następujące problemy:

ograniczenie miejsca na dysku Pomimo, że przestrzeń dyskowa dla komputerów tanieje, trzymanie kilkugigabajtowego klucza może być w niektórych przypadkach niewykonalne, np. dla urządzeń mobilnych (smartfonów, tabletów), których pamięć jest ograniczona w sposób znaczenie bardziej restrykcyjny niż pamięć komputerów i nie może być łatwo powiększona.

spowolnienie obliczeń Obliczenia na kilkugigabajtowym kluczu ciężko wykonać w podobnym czasie, co obliczenia dla klucza kilkukilobajtowego. Już samo wczytanie tak dużych danych do pamięci zajmuje czas, nie wspominając już o zasobach systemowych. Dlatego też protokoły BRM działają w czasie zależnym od parametru bezpieczeństwa, i co najwyżej wielomianowym od logarytmu długości klucza. To bardzo ważne założenie sprawia, że nie jest dopuszczalnym korzystanie z całego klucza a jedynie z jego części. W związku z tym większość protokołów BRM korzysta z losowych (ze świeżą losowością przy każdym użyciu) bitów klucza.

1.2.1 Losowa wyrocznia

W pracy wykorzystujemy model z losową wyrocznią. Losowa wyrocznia Ω może być zdefiniowana następująco. Mając dwa zbiory $\mathbf{X}$, $\mathbf{Y}$ (zwykle wymagamy by $\mathbf{Y}$ był skończony) wyrocznia $\Omega_{\mathbf{X},\mathbf{Y}}$ losuje funkcję $\mathcal{H} : \mathbf{X} \to \mathbf{Y}$, która na zapytanie $x \in \mathbf{X}$ odpowiada $y = \mathcal{H}(x)$. W implementacjach losowa wyrocznia jest zastępowana funkcją haszującą. W dalszej części pracy utożsamiamy losową wyrocznię z funkcją przez nią wylosowaną tj. $\mathcal{H}$.

Canetti i in. [5] i Nielsen [34] pokazali (niezależnie), że żadna funkcja deterministyczna nie może być losową wyrocznią. Ich dowody opierały się na wskazaniu konstrukcji, których bezpieczeństwa można dowieść w modelu z losową wyrocznią, ale nie w modelu standardowym.

Pomimo tych wyników, model z losową wyrocznią jest dość powszechnie stosowany i ma swoich obrońców. Jak wskazują Menezes i Koblitz w [28], konstrukcje Nielsena, Canettiego i in. są oparte na konstrukcjach, które są "nienaturalne" i nie miałyby szans na zastosowanie w praktyce. Wskazują ponadto, że losowa wyrocznia bywa zastępowana innymi założeniami, dużo mniej poznanymi i dużo bardziej skomplikowanymi. Ponadto konstrukcje używające losowej wyroczni są zwykle prostsze, co ma istotne znaczenie przy implementacji—jak zwracaliśmy uwagę wcześniej, błędy w implementacji mogą powodować bardzo duże zagrożenie dla bezpieczeństwa systemu.

2 Wyniki rozprawy

Wyniki rozprawy zostały w dużej części opublikowane w artykule Konrada Durnogi, Stefana Dziembowskiego, Tomasza Kazany, Michała Zająca i Macieja Zdanowicza *Bounded-Retrieval Model with Keys Derived from Private Data* [12].

2.1 Wydajna metoda uzyskiwania klucza prywatnego w modelu BRM

W rozprawie pokazujemy jak uzyskać bezpieczny i wydajny pamięciowo schemat BRM. Jak zostało wspomniane wyżej, użytkownik chcący zapewnić sobie bezpieczeństwo w modelu BRM jest często zmuszony do trzymania w pamięci swojego komputera gigantycznego losowego klucza, który nie ma pozakryptograficznych zastosowań. To sprawia, że model ten nie nadaje się do zastosowania w urządzeniach mobilnych, których pamięć jest mocno ograniczona i nie daje się jej łatwo powiększyć.

Pokażemy jak zamiast klucza kilkugigabajtowego klucza, będącego próbką zmiennej losowej o jednostajnym rozkładzie, uzyskać bezpieczny klucz ze zmiennej o rozkładzie niejednostajnym. Podobnie jak Dodis i in. [11], procedurę pozyskiwania klucza z danych niejednostajnych nazywamy *key derivation function* (kdf). W odróżnieniu jednak od [11], nasza kdf ma istotną zaletę. Mianowicie, pozwala ona na uzyskanie bezpiecznego klucza kryptograficznego z prywatnych danych użytkownika.

2.1.1 Klucz kryptograficzny z prywatnych danych użytkownika

Pozyskiwanie klucza kryptograficznego do modelu BRM z danych dyskowych nie jest problemem trywialnym. Istotne jest, że w pewnych okolicznościach część klucza może zostać poznana przez osoby trzecie. Wynika to z dwóch czynników.

- po pierwsze, przeciwnik $\mathcal{A}$ jest wyposażony w wyciek,
- po drugie, niektóre protokoły w BRM zdradzają część klucza gdy są wykonywane.

Punkt pierwszy wynika z samej istoty modelu, więc nie będziemy się w niego zagłębiać. Punkt drugi wymaga krótkiego wyjaśnienia. Dla przykładu rozważmy protokół identyfikacji w modelu BRM. W protokole tym biorą udział dowodzący $\mathcal{P}$ i sprawdzający $\mathcal{V}$. Dowodzący $\mathcal{P}$ posiada tajny klucz prywatny (duży), sprawdzający $\mathcal{V}$ posiada klucz weryfikujący (mały, może to być, np. klucz publiczny dowodzącego). Niektóre protokoły pozwalają by weryfikujący poznał część klucza prywatnego dowodzącego $\mathcal{P}$. Nie stanowi to jednak zagrożenia dla bezpieczeństwa systemu – poznane fragmenty klucza są albo zbyt małe, by stanowiły jakąkolwiek wartość, albo są wykorzystywane jednorazowo – raz użyte do identyfikacji nie mogą zostać użyte ponownie.

Obydwa czynniki stanowią istotne zagrożenie dla prywatności danych, które zostały użyte do wygenerowania klucza kryptograficznego. O ile w przypadku wycieku możliwości ochrony są ograniczone – przeciwnik może również wyciec informacje znajdujące się bezpośrednio na dysku, o tyle w przypadku używania klucza zgodnie z przeznaczeniem ochrona prywatności danych jest koniecznością. Rozwiązanie, w którym weryfikujący $\mathcal{V}$ jest w stanie poznać prywatne dane użytkownika podczas wykonywania protokołu nie może zostać zaakceptowane. Dlatego też stawiamy przed kdf bardzo istotne wymaganie – poza wyciekiem, klucz nie może zdradzać żadnych informacji o danych.

2.1.2 Jakie dane są prywatne?

Przed przystąpieniem do generowania klucza z danych prywatnych należy odpowiedzieć na pytanie z jakich danych możemy wygenerować bezpieczny klucz, a z jakich nie. Odpowiedź na to pytanie nie jest tak jednoznaczna, jak może się wydawać.

Na przykład, załóżmy, że chcemy użyć do wygenerowania klucza pewnego odcinka naszego ulubionego serialu, który trzymamy na dysku. W takim przypadku ciężko mówić o tym, że dane te są jakkolwiek losowe. Przeciwnik, dysponując bardzo małym wyciekiem, może się dowiedzieć jaki to serial, który sezon i odcinek. Zauważmy, te informacje nie są tajne i są ponadto zawarte w metadanych pliku. Mając te dane, łatwo może znaleźć w internecie identyczny plik. Nawet jeśli ten sam plik istnieje w różnych wersjach, ich liczba jest zbyt mała by uznać generowanie klucza prywatnego z powszechnie dostępnych multimediów za bezpieczne.

W związku z powyższym, pozwalamy użytkownikowi na wygenerowanie klucza z danych prywatnych, ale takich, których on sam jest autorem a które nie są publicznie dostępne. Należy zadać pytanie, jak wiele jest prywatnych danych, które nie są publicznie dostępne w dobie wszelkiego rodzaju portali społecznościowych. Zakładamy tu następujące dwie rzeczy. Po pierwsze, ze względu na ograniczoną przestrzeń dyskową serwerów portali społecznościowych multimedia tam wgrywane są kompresowane. Zatem, mimo że zdjęcie na telefonie przedstawia to samo co zdjęcie np. na Facebooku, są to w istocie różne pliki. Po drugie, zakładamy, że nie wszystkimi danymi użytkownik się dzieli z każdym. To jest, częścią danych użytkownik dzieli się tylko w ograniczonym gronie przyjaciół.

Czy w przypadku, w którym dane prywatne, z których można wygenerować klucz są ograniczone tylko do danych stworzonych przez użytkownika możemy liczyć na odpowiednią ich ilość? Uważamy, że tak. Stwierdzenie to popieramy dwoma statystykami. Po pierwsze, w każdej minucie 300 godzin filmów jest zamieszczanych na Youtube [4]. Po drugie, każdego dnia ponad 1,8 miliarda zdjęć trafia na Facebooka, Instagram i Snapchat [18]. To pokazuje jak bardzo kreatywni są ludzie i ile treści tworzą.

2.1.3 Lokalność obliczeń

Chcemy podkreślić, że zaproponowana metoda uzyskiwania klucza pozwala na obliczenie dowolnej jego części bez obliczania klucza w całości. Jest to własność szczególnie istotna w schematach BRM, w których klucz jest duży. Ponadto, protokoły kryptograficzne w BRM korzystają tylko z pewnego fragmentu klucza, zatem każdorazowe obliczanie całości klucza byłoby niewątpliwie stratą czasu i trwonieniem mocy obliczeniowej. Co więcej, obliczenia mogą być przeprowadzone na bieżąco z danych dyskowych, nie ma potrzeby trzymania jakiejkolwiek informacji dodatkowych.

2.2 Key Derivation Functions (kdf)

Jak wspomniano wcześniej, idea *key derivation functions* została zaproponowana przez Dodisa i in. w [11]. Tutaj wprowadzamy formalną definicję tej klasy. Funkcje kdf przekształcają zmienną X o nieidealnej min-entropii w bezpieczny klucz kryptograficzny. Zgodnie z tym co proponowaliśmy powyżej, stwierdzamy, że prywatne dane użytkownika przechowywane na urządzeniu mogą posłużyć jako X.

Z takim podejściem wiąże się jednak duże niebezpieczeństwo ujawnienia części prywatnych danych. Wynika to z faktu, że niektóre protokoły w modelu BRM ujawniają część klucza podczas np. identyfikacji. Dlatego też wymagamy by funkcje kdf były nie tylko *bezpieczne*, ale i *prywatne*. Obydwa wyrażenia zostały zdefiniowane poniżej.

2.2.1 Bezpieczeństwo

Kluczową własnością funkcji kdf jest jej bezpieczeństwo. Zanim przejdziemy do formalnej definicji tego terminu przyjmiemy, że funkcja kdf jest bezpieczna, kiedy różnica pomiędzy bezpieczeństwem protokołu w przypadku kiedy klucz kryptograficzny jest jednostajnie losowy, a bezpieczeństwem, kiedy ten sam protokół korzysta z klucza wygenerowanego przez kdf jest zaniedbywalnie mała.

Do sformalizowania definicji bezpieczeństwa potrzebne jest wprowadzenie tzw. *gier bezpieczeństwa*. Gry bezpieczeństwa to to gry, w których uczestniczą dwie strony – przeciwnik $\mathcal{A}$ i testujący $\mathcal{C}$. Testujący odpowiada na pytania przeciwnika (tzw. *faza nauki*), a w końcowej fazie gry (tzw. *faza testu*) zadaje mu pytanie. Jeśli $\mathcal{A}$ odpowie prawidłowo, uznajemy że wygrał. Bezpieczeństwo większości protokołów czy schematów kryptograficznych jest zdefiniowane właśnie za pomocą ww. gier. Jako przykład posłuży gra definiująca bezpieczeństwo schematu szyfrowania (IND-CPA).

Przykład 3 (Gra bezpieczeństwa IND-CPA dla schematu szyfrowania, zob. [27])**.** *Gra przebiega następująco:*

Faza przygotowawcza: *Niech* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *będzie schematem szyfrowania.* $\mathsf{Enc}, \mathsf{Dec}$ *są parą funkcji szyfrującej i deszyfrującej. Para klucza prywatnego i publicznego* $(\mathtt{sk}, \mathtt{pk})$ *została utworzona za pomocą algorytmu* KeyGen *uruchomionego na jednostajnie losowej zmiennej* R *oraz parametrze bezpieczeństwa* n *(podanym unarnie). Oznaczymy przez* $\mathcal{E}^{\mathtt{pk}}$ *wyrocznię szyfrującą. Testujący* $\mathcal{C}$ *obsługuje wyrocznię* $\mathcal{E}^{\mathtt{pk}}$*. Przeciwnik* $\mathcal{A}$ *posiada dostęp do wyroczni* $\mathcal{E}^{\mathtt{pk}}$*, klucz publiczny* pk *oraz inne publiczne parametry schematu szyfrowania.*

Faza nauki: *Przeciwnik* $\mathcal{A}$ *może wysłać do wyroczni szyfrującej wielomianowo wiele wiadomości* m_i *i uzyskać od* $\mathcal{E}^{\mathtt{pk}}$ *szyfrogramy* $\mathsf{Enc}_{\mathtt{pk}}(m_i)$*. Przeciwnik jest adaptywny, tj. może wybrać wiadomość* m_i *uwzględniając wcześniejsze* $(m_{i'}, \mathsf{Enc}_{\mathtt{pk}}(m_{i'}))$*, dla* $i' < i$*.*

Faza testu: *Następnie,* $\mathcal{A}$ *wybiera parę wiadomości* m_0 *oraz* m_1 *i wysyła je do* $\mathcal{E}$*. Wyrocznia* $\mathcal{E}$ *wybiera losowy bit* b *i odpowiada z* $\mathsf{Enc}_{\mathtt{pk}}(m_b)$*. Przeciwnik* $\mathcal{A}$ *zgaduje jaki bit* b *został wylosowany przez* $\mathcal{E}$ *i daje na wyjściu bit* b'*.*

Mówimy, że przeciwnik wygrywa grę jeżeli $b = b'$*. Mówimy, że schemat jest* $\varepsilon(n)$*-bezpieczny w rozumieniu tej gry jeżeli prawdopodobieństwo wygrania przeciwnika* $\mathcal{A}$ *jest nie większe niż* $\frac{1}{2} + \varepsilon(n)$

Uwaga. Zauważmy, nie wszystkie gry polegają na rozróżnieniu dwóch przypadków (zgadywaniu wylosowanego bitu). W niektórych przypadkach bezpieczeństwo wynosi $\varepsilon(n)$, jeżeli prawdopodobieństwo zwycięstwa $\mathcal{A}$ wynosi $\varepsilon(n)$.

Mówimy, że gra Game jest $(\varepsilon(n), \mathbf{TM})$-bezpieczna, jeżeli dla każdego algorytmu $\mathcal{A} \in \mathbf{TM}$,

$$\Pr(\mathcal{A} \text{ wygrywa } \mathsf{Game}) \leqslant \varepsilon(n) \ ,$$

dla parametru bezpieczeństwa n.

Zwykle klasa **TM** będzie jasno wynikała z kontekstu. Dla ww. gry **TM** to klasa wszystkich probabilistycznych algorytmów działających w czasie wielomianowym. W dalszej części przez $\mathbf{TM}^{\mathcal{O}(D,\mathcal{H}),\mathcal{H}}_{\lambda,q(n)}$ oznaczamy wszystkie algorytmy, które

- mają dostęp do wyroczni wyciekowej $\mathcal{O}(D, \mathcal{H})$ ograniczony parametrem wycieku λ, oraz
- mogą zadać co najwyżej $q(n)$ zapytań do wyroczni losowej $\mathcal{H}$.

W szczególności, nie nakładamy na algorytmy z $\mathbf{TM}^{\mathcal{O}(D,\mathcal{H}),\mathcal{H}}_{\lambda,q(n)}$ ograniczeń dot. złożoności obliczeniowej. Poniżej przedstawiamy (jeszcze nieformalnie) definicję bezpieczeństwa kdf.

Definicja 1 (Bezpieczeństwo kdf)**.** Rozważmy protokół Π, generujący klucz za pomocą algorytmu KeyGen działającego na danych jednostajnie losowych. Niech Game będzie grą bezpieczeństwa dla Π, taką że przeciwnik $\mathcal{A}$ wygrywa Game z prawdopodobieństwem nie większym niż $\varepsilon(n)$, dla parametru bezpieczeństwa n. Mówimy, że kdf jest $\varepsilon'(n)$ bezpieczna jeżeli po zamienieniu w Π algorytmu KeyGen na kdf i jednostajnie losowych danych na dane z niejednostajnego rozkładu, przeciwnik $\mathcal{A}$ wygrywa w Game z prawdopodobieństwem nie większym niż $\varepsilon(n) + \varepsilon'(n)$.

Kluczowym dla definicji bezpieczeństwa funkcji kdf jest fakt, że pasuje ona do każdego protokołu, którego bezpieczeństwo jest zdefiniowane za pomocą gry a klucz KeyGen może być wygenerowany lokalnie, np. przez testującego $\mathcal{C}$. Czyli do większości protokołów. Po zastosowaniu kdf nie zmienia się definicja bezpieczeństwa protokołu, jedynie sposób generowania klucza zostaje zmieniony.

2.2.2 Prywatność

Kolejną kluczową kwestią związaną z funkcjami kdf dla schematów w modelu BRM jest zachowanie prywatności danych, z których uzyskiwany jest klucz.

Prywatność definiujemy następująco. Rozważmy dowolny probabilistyczny algorytm $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}(D,\mathcal{H}),\mathcal{H}}_{\lambda,q(n)}$. Powiedzmy, że algorytm $\mathcal{A}$ dostaje na wejściu klucz K otrzymany za pomocą funkcji kdf działającej na zmiennej losowej D. Dla takiego $\mathcal{A}$ konstruujemy symulator $\mathcal{S}$, który dostaje opis $\mathcal{A}$ na wejściu (np. jako opis maszyny Turinga). Co istotne, $\mathcal{S}$ nie dostaje klucza K ani innych danych wejściowych $\mathcal{A}$. Mówimy, że funkcja kdf jest bezpieczna, jeżeli wyjście symulatora i wyjście algorytmu $\mathcal{A}$ leżą $\varepsilon(n)$-blisko w sensie dystansu statystycznego Δ nawet wtedy, gdy znana jest zmienna D. Formalnie możemy tę własność zapisać następująco:

$$\Delta((\mathbf{Output}(\mathcal{A}(K)), D), (\mathbf{Output}(\mathcal{S}(\mathcal{A})), D)) \leq \varepsilon(n) \ ,$$

gdzie dystans statystyczny Δ pomiędzy zmiennymi X i Y jest zdefiniowany jako

$$\Delta(X, Y) = \frac{1}{2} \sum_{a \in \mathrm{supp}(X) \cup \mathrm{supp}(Y)} |\Pr(X = a) - \Pr(Y = a)| \ .$$

Idea tego przedstawienia jest następująca: skoro wyjście $\mathcal{A}$ mającego dostęp do klucza K jest nieodróżnialne od wyjścia symulatora $\mathcal{S}$ dostępu do tego klucza pozbawionego, to algorytm $\mathcal{A}$

nie jest w stanie uzyskać żadnych znaczących danych na temat zmiennej D. Zauważmy, gdyby $\mathcal{A}$ mógł uzyskać jakiekolwiek dane o D, to mógłby je przekazać na swoje wyjście. Symulator nie jest w stanie zrobić podobnej rzeczy, ponieważ nie ma dostępu do D. Obserwator wyposażony w dostęp do wyjść $\mathcal{A}$ i $\mathcal{S}$, jak i w D łatwo mógłby rozróżnić $\mathcal{A}$ od $\mathcal{S}$ porównując wyjścia z informacjami zawartymi w D.

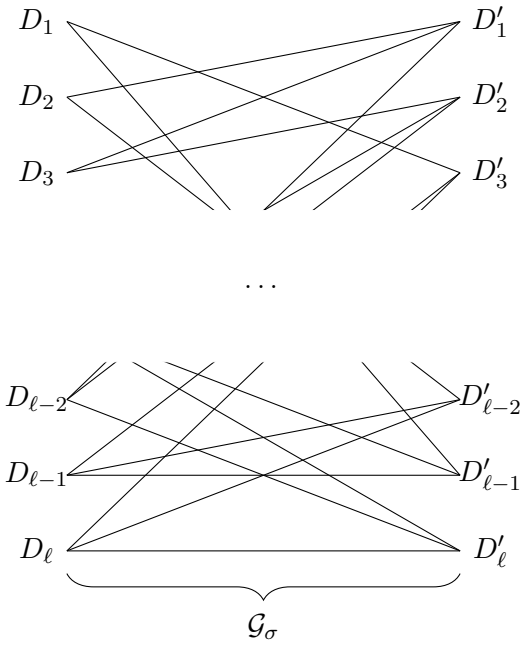
2.3 Disperse jako przykład kdf

W rozprawie pokazujemy konkretny przykład bezpiecznej i prywatnej funkcji kdf. Funkcja ta jest zbudowana z użyciem dwudzielnego prawego d-regularnego grafu będącego disperserem (stąd też nazwa naszej funkcji—Disperse). Graf $(\mathbf{V}_0 \sqcup \mathbf{V}_1, E)$ nazywamy dwudzielnym prawo d-regularnym (K, ε)-disperserem jeśli jest

1. dwudzielny;
2. każdy jego wierzchołek z prawej stronie ma stopień dokładnie d;
3. każdy zbiór prawych wierzchołków o liczności co najmniej K jest połączony z co najmniej $(1-\varepsilon)|\mathbf{V}_0|$ wierzchołkami z lewej strony.

Dla ustalenia uwagi przyjmiemy, że po lewej stronie grafu znajdują się wierzchołki o wartościach będącymi kolejnymi blokami próbki zmiennej losowej D; a po prawej kolejne bloki klucza (ozn. D'). j-ty blok klucza, tj. D'_j ma wartość $\mathcal{H}(j, D_{\sigma(j,1)}, \ldots, D_{\sigma(j,\ell^d)})$, gdzie $\mathcal{H}$ to losowa wyrocznia, a σ funkcja opisująca graf $\mathcal{G}_\sigma$. Przyjmujemy, że zarówno klucz jak i dane, z których klucz jest generowany są podzielone na ℓ bloków długości n, tj. $|\mathbf{V}_0| = |\mathbf{V}_1| = \ell$.

Przykładowa funkcja Disperse została zaprezentowana na Rysunku 1.



Rysunek 1: Przykładowy $\mathsf{Disperse}_{\mathcal{G}_\sigma}(D, \mathcal{H})$

2.3.1 Jednokierunkowość

Jedną ze szczególnie istotnych właściwości naszej konstrukcji przedstawionej powyżej jest tzw. *jednokierunkowość*. Jednokierunkowość jest kluczowa zarówno dla prywatności, jak i bezpieczeństwa funkcji Disperse. Nieformalnie możemy tę własność opisać następująco. Jeżeli przeciwnik $\mathcal{A}$ ma dostęp do wartości wszystkich prawych wierzchołków D'_j grafu $\mathcal{G}_\sigma$ (tj. ma dostęp do całego klucza), ale jednocześnie jego dostęp do lewych wierzchołków D_i jest ograniczony przez parametr wycieku λ, to liczba zapytań do wyroczni o wartości postaci $\mathcal{H}(i, D_{\sigma(i,1)}, \ldots, D_{\sigma(i,\ell^d)})$, dla $i = 1, \ldots, \ell$, jest ściśle ograniczona poza pewnym zaniedbywalnym prawdopodobieństwem.

2.4 Schematy identyfikacji i podpisu cyfrowego

Poza podaniem przykładu funkcji kdf pokazujemy przykładowe protokoły identyfikacji i podpisu cyfrowego, działające w modelu BRM, w których procedurę generowania klucza można zastąpić funkcją Disperse. Obydwie konstrukcje są otrzymane za pomocą drzew Merkla, które gwarantują efektywność obliczeń i mały rozmiar klucza publicznego. Zauważmy, że mały rozmiar klucza publicznego jest konieczny by móc stosować schematy BRM w szerszej skali, gdyż każda ze stron uczestnicząca w protokole musi trzymać klucze publiczne każdej innej strony. W przypadku, kiedy klucz publiczny miałby długość porównywalną do klucza prywatnego, tj. byłby rzędu kilku gigabajtów, każda maszyna musiałaby przechowywać wiele kilkugigabajtowych kluczy. W zaproponowanym rozwiązaniu klucz publiczny jest długości jednego bloku klucza prywatnego, tj. n.

Schemat podpisu cyfrowego powstał poprzez zastosowanie heurystyki Fiata-Shamira [20, 19] do schematu identyfikacji (co było możliwe dzięki temu, że działamy w modelu z losową wyrocznią).

2.5 Podpis cyfrowy w modelu BRM

W modelu BRM problem podpisu cyfrowego ma szczególną postać. Jak argumentowaliśmy wcześniej, parametr wycieku ma (dla praktyczności modelu) wielkość liczoną w gigabajtach danych. Podpis cyfrowy jest zwykle zdecydowanie krótszy, długości, np. kilku kilobajtów. Przeciwnik zatem może postąpić następująco: Załóżmy użytkownika z kluczami $\mathtt{pk}, \mathtt{sk}$, gdzie $\mathtt{sk}$ jest tajnym kluczem służącym do podpisywania wiadomości, a $\mathtt{pk}$ kluczem publicznym do weryfikowania podpisu. Chcąc uzyskać fałyszywy podpis cyfrowy pod dokumentem m, przeciwnik $\mathcal{A}$ konstruuje funkcję wycieku f w ten sposób, że $f(\mathtt{sk})$ jest prawdziwym podpisem cyfrowym na m. Jest to możliwe ponieważ $f(\mathtt{sk}) \ll \lambda$.

Powyższy przykład wskazuje, że nie jest możliwe osiągnięcie *standardowej* odporności na fałszerstwo (ang. *existential unforgeability*, dosłownie niefałszowalność egzystencjalna) — nie można zagwarantować tego, że przeciwnik nie będzie mógł (z niezaniedbywalnym prawdopodobieństwem) wytworzyć podpisu dla żadnej wiadomości. Dlatego też [2] zaproponowali nowy rodzaj przeciwnika, przeciwnika entropijnego. Mówimy, że schemat podpisu cyfrowego jest entropijnie niefałszowlany (ang. *entropic unforgeability*, dosłownie niefałszowalność entropijna), jeżeli jest bezpieczny dla entropijnych przeciwników. Intuicyjne definicje podano poniżej.

Entropijny przeciwnik $\mathcal{A}$ jest zdefiniowany następująco: $\mathcal{A}$ jest podzielony na dwa algorytmy $\mathcal{A}_1, \mathcal{A}_2$. Algorytm $\mathcal{A}_1$ może wykonywać zapytania do wyroczni wyciekowej i do wyroczni podpisującej. Może także przekazać wskazówkę *hint* do $\mathcal{A}_2$. Algorytm $\mathcal{A}_2$ nie ma dostępu do wyroczni wyciekowej, może za to zadawać zapytania do wyroczni podpisującej. Przeciwnik $\mathcal{A}$ jest entropijny, min-entropia zmiennej losowej M, z której $\mathcal{A}_2$ wybiera podpisywaną wiadomość ma min-entropię co najmniej n z punktu widzenia $\mathcal{A}_1$ (tj. uwzględniając wszystkie informacje dostępne dla $\mathcal{A}_1$).

Mówimy, że schemat podpisu jest entropijnie niefałszowalny, jeżeli jest bezpieczny przeciwko entropijnym przeciwnikom.

2.5.1 Od identyfikacji do podpisu cyfrowego

Klasyczna konstrukcja Fiata-Shamira pozwala na zbudowanie schematu podpisu cyfrowego z protokołu identyfikującego. Heurystyka Fiata-Shamira pozwala na przekształcenie dowolnego protokołu identyfikującego postaci (a, c, z), gdzie a jest pierwszą wiadomością wysłana przez identyfikującego się, c pytaniem przysłanym przez weryfikującego (c jest otrzymane z rozkładu jednostajnego), a z odpowiedzią na c uwzględniającą a, w protokół nieinteraktywny, przez zastąpienie c wartością $\mathcal{H}(a)$, dla wyroczni losowej $\mathcal{H}$. Przekształcając protokół identyfikacji w protokół podpisu cyfrowego wystarczy zastąpić pierwszą wiadomość a, podpisywaną wiadomością m. W takim przypadku, podpisem jest trójka $(m, \mathcal{H}(m), z)$.

Alwen i in. [2] pokazali, że za pomocą heurystyki Fiata-Shamira można także przekształcić schemat identyfikacji w modelu BRM w entropijnie bezpieczny schemat podpisu cyfrowego w tymże modelu. W rozprawie pokazujemy, że ten wynik działa również w przypadku, kiedy działamy w modelu z ograniczoną liczbą zapytań.

2.5.2 Ograniczona liczba zapytań

Zazwyczaj, kiedy dajemy przeciwnikowi $\mathcal{A}$ dostęp do wyroczni podpisującej, czy też identyfikującej, $\mathcal{A}$ może im zadań wielomianową (od długości parametru bezpieczeństwa) liczbę zapytań. Niestety, w przypadku modelu BRM, część klucza prywatnego może zostać "zużyta" podczas wykonywania na nim czynności. Tj. klucz prywatny, po pewnym czasie, przestanie być bezpieczny. Dlatego też w tej rozprawie używamy modelu, w którym przeciwnik ma ograniczoną z góry liczbę zapytań do wyroczni. Pokazujemy także, że dla parametrów wziętych z życia, przyjęte ograniczenie nie jest dotkliwe, a ponadto klucz można *odświeżyć*.

2.6 Odświeżanie klucza

Jak zostało wcześniej wspomniane, jedną z głównych motywacji dla tej pracy było stworzenie praktycznego schematu identyfikacji w modelu BRM. Praktyczność uzyskaliśmy poprzez generowanie klucza prywatnego za pomocą funkcji kdf działającej na pewnej zmiennej D o niedoskonałej losowości. Argumentowaliśmy, że jako ową zmienną D możemy traktować prywatne dane dyskowe użytkownika schematu.

Z wykorzystywaniem danych dyskowych jako klucza prywatnego pojawia się następujący problem. Klucz zostaje zmieniony ilekroć zmieniają się dane, z których jest on otrzymywany. Znacząco ogranicza to jego możliwości. Rozwiązanie, które zakładałoby, że dane dyskowe pozostają niezmienne nie byłoby optymalne. Dlatego też konieczne było dostarczenie protokołu, który w szybki i efektywny sposób zaktualizuje klucz. To jest, dla zmienionego klucza prywatnego zaktualizuje klucz publiczny. W rozprawie przedstawiamy metodę odświeżenia klucza publicznego dla schematów identyfikacji i podpisu cyfrowego. Odświeżenie klucza wymaga wysłania tylko jednej wiadomości.

2.7 Inne podejścia do rozwiązania przedstawionego problemu

Mogłoby się wydawać, że przedstawiony problem uzyskania klucza kryptograficznego ze zmiennej o niejednostajnym rozkładzie może być rozwiązany za pomocą prostszych metod. W tej części pokazujemy, że pewne proste i "naturalne" podejścia do problemu nie działają lub nie spełniają wszystkich wymagań.

Ekstraktory losowości. Zaczniemy od rozwiązania opartego na ekstraktorach losowości zamiast wyroczni losowej [22, 9, 3]. Funkcję Ext nazwiemy (k, ε)-ekstraktorem, jeżeli dla pewnej zmiennej losowej R o rozkładzie jednostajnie losowym nad $\{0,1\}^d$ i zmiennej $X \in \{0,1\}^n$, takiej że $\widetilde{\mathrm{H}}_\infty(X \mid L) \geqslant k$, gdzie L to dodatkowa informacja o X, $\mathrm{Ext}(X,R) \in \{0,1\}^m$ takie, że

$$\Delta((U_m, R, L), (\mathrm{Ext}(X,R), R, L)) \leqslant \varepsilon \ ,$$

dla U_m zmiennej losowej o rozkładzie jednostajnym nad $\{0,1\}^m$.

W przedstawionym problemie, wartość zmiennej R powinna być przechowywana na dysku, co wiąże się m.in. z tym, że przeciwnik $\mathcal{A}$ może uzyskać pewne informacje na jej temat. W szczególności wyciek L może być zależny także od R. Jak łatwo zauważyć, wyżej wymieniona definicja nie uwzględnia takiego przypadku – wyciek L może zależeć tylko od X.

Ponadto, ze względu na rozmiar klucza, wymagamy od funkcji kdf lokalności, jak zdefiniowano powyżej. Niestety, znane ekstraktory nie posiadają takiej właściwości – wynik jest obliczany w całości, albo w ogóle.

Haszowanie blok po bloku. Kolejnym elementem naszego rozwiązania są dispersery [22, 24, 35]. Pokażemy teraz dlaczego grafy te są istotnym elementem naszego rozwiązania.

Załóżmy, że nasze dane D składają się z bloków $D_1, \ldots, D_\ell$. Zamiast przykładać do tych danych funkcję wyznaczoną przez pewien disperser, moglibyśmy wywoływać losową wyrocznię na każdym z tych bloków niezależnie otrzymując klucz $D'_1 = \mathcal{H}(1, D_1), \ldots, D'_\ell = \mathcal{H}(\ell, D_\ell)$, co niewątpliwie byłoby znacznie mniej złożone obliczeniowo niż zaproponowane w pracy rozwiązanie.

Niestety, w naszym problemie nie możemy przyjąć, że min-entropia jest "rozdzielona" pomiędzy bloki $D_1, \ldots, D_\ell$ równomiernie. Pomiędzy blokami D mogą znajdować się bloki całkowicie nielosowe, takie jak np. ustalone nagłówki plików o min-entropii bliskiej zera. W takim przypadku podczas wywoływania losowej wyroczni blok po bloku, przeciwnik może poznać pewną dodatkową informację o danych znajdujących się u podstawy klucza.

Ponadto niektóre bloki mogą być takie same, co pozwala przeciwnikowi na uzyskanie informacji o dwóch blokach kosztem wycieku jednego z nich. Stąd uzasadnienie dla rozwiązania, w którym każdy blok klucza jest zależny od wielu bloków wejściowych i stąd uzasadnienie dla wykorzystania w rozwiązaniu disperserów.

3 Key Derivation Function

W tej części przedstawiamy jeden z głównych wyników rozprawy. Tak jak zostało wspomniane wcześniej, jedną z głównych wad modelu BRM jest jego nieunikniona niewydajność pod kątem przestrzeni zajmowanej na dysku. Nasze rozwiązanie pozwala na zaoszczędzenie przestrzeni dyskowej potrzebnej do zapewnienia bezpieczeństwa wykonywania protokołu BRM. Zamiast przechowywać na urządzeniu całkowicie losowy klucz, który nie ma pozakryptograficznych zastosowań,

będziemy generować klucz z danych znajdujących się na dysku. W szczególności, nie będzie potrzeby przechowywania klucza na urządzeniu, potrzebne jego fragmenty będą generowane na bieżąco.

Od tego miejsca oznaczymy przez D zmienną losową reprezentującą dane dyskowe długości N (wszystkie długości wyrażamy dla zapisu binarnego), przez λ maksymalną ilość wycieku dostępnego adwersarzowi, M długością wygenerowanego klucza, a p proporcją min-entropii danych D i N, tj. $\mathrm{H}_\infty(D) = pN$. Mówimy, że przeciwnik $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}(D,\mathcal{H}),\mathcal{H}}_{\lambda,q(n)}$ jeżeli $\mathcal{A}$ jest maszyną Turinga posiadającą dostęp do wyroczni wyciekowej $\mathcal{O}$, którą może pytać o dane z D i $\mathcal{H}$, i losowej $\mathcal{H}$, przy czym maksymalny rozmiar wycieku jest ograniczony przez λ a liczba zapytań do wyroczni $\mathcal{H}$ przez $q(n)$ (ponieważ dla każdego algorytmu w pracy zakładamy ograniczenie na $q(n)$ zapytań do wyroczni losowej, a jedynym istotnym parametrem jest rozmiar wycieku λ, będziemy czasem pomijać parametr $q(n)$ w zapisie).

3.1 Bezpieczeństwo i prywatność kdf

Dwiema najważniejszymi właściwościami kdf są bezpieczeństwo i prywatność. Pierwsza odpowiada za zaniedbywalną różnicę w bezpieczeństwie pomiędzy protokołem uruchomionym na losowym kluczu, a uruchomionym na kluczu uzyskanym za pomocą kdf. Druga jest o tyle istotna, że nie jest sytuacją dopuszczalną by prywatne dane użyte w kdf, stały się danymi publicznymi.

Definicja 2 (Prywatność w kdf)**.** Mówimy, że probabilistyczna funkcja $\mathsf{kdf}_\mathcal{H} : \mathbb{N} \times \{0,1\}^N \to \{0,1\}^M$ jest $(\Lambda(\lambda), q(n), \varepsilon(n))$-prywatna jeśli istnieje symulator $\mathcal{S} \in \mathbf{TM}^{\mathcal{O}(D),\mathcal{H}}_{\lambda+\Delta_\lambda,q(n)}$, jeśli dla każdej zmiennej losowej $D \in \{0,1\}^N$ o min-entropii $\mathrm{H}_\infty(D) \geqslant \kappa$ i każdego algorytmu $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}(D,\mathcal{H}),\mathcal{H}}_{\lambda,q(n)}$ działającego na $K = \mathsf{kdf}_\mathcal{H}(D)$ zachodzi

$$\Delta((\mathbf{Output}(\mathcal{A}(K)), D), (\mathbf{Output}(\mathcal{S}(\mathcal{A})), D)) \leq \varepsilon(n) \ .$$

Definicja 3 (Bezpieczeństwo kdf)**.** Mówimy, że probabilistyczna funkcja $\mathsf{kdf}_\mathcal{H} : \{0,1\}^N \to \{0,1\}^M$ jest $(\Lambda(\lambda), q(n), \varepsilon'(n))$-bezpieczna jeżeli dla każdego $(\varepsilon(n), \mathbf{TM}^{\mathcal{O}(K)}_\lambda)$-bezpiecznej gry

$$\mathsf{Game} = (\mathcal{C}, \mathsf{KeyGen}(1^n; R), \mathsf{Setup}_{\mathrm{Ch}}, \mathsf{Setup}_{\mathrm{Adv}}, \mathsf{Execute}) \ ,$$

gdzie R jest zmienną losową o rozkładzie jednostajnym nad $\{0,1\}^N$, gra

$$\mathsf{Game}_{\mathrm{Disk}} = (\mathcal{C}, \mathsf{KeyGen}(1^n; \mathsf{kdf}_\mathcal{H}(1^n; D)), \mathsf{Setup}_{\mathrm{Ch}}, \mathsf{Setup}_{\mathrm{Adv}}, \mathsf{Execute})$$

na losowości $(D, \mathcal{H})$ jest $(\varepsilon'(n) + \varepsilon(n), \mathbf{TM}^{\mathcal{O}(D,\mathcal{H}),\mathcal{H}}_{\lambda-\Delta_\lambda,q(n)})$-bezpieczna.

Definicja prywatności definiuje jakiej wielkości dodatkowy wyciek Δ_λ jest potrzebny symulatorowi by wynikowa zmienna losowa miała pożądaną własność nieodróżnialności. Zwróćmy uwagę, że w tym modelu symulator ma minimalnie większe możliwości niż adwersarz. Oczywiście, chcemy by różnica możliwości była jak najmniejsza, dlatego też chcemy by jak najmniejszego Δ_λ. Definicja bezpieczeństwa zapewnia, że adwersarz nie zyskuje znaczącej przewagi kiedy w użyciu jest klucz kdf zamiast losowego.

Zauważymy, definicja 3 używa kdf do wygenerowania *losowości* dla algorytmu generującego klucz KeyGen, nie samego klucza. W dalszej części pracy przyjmiemy uproszczoną wersję algorytmu KeyGen, tj. $\mathsf{KeyGen}(1^n; R) = R$. (Dlatego też nazywamy naszą funkcję *key derivation function* a nie *randomness derivation function*, czy podobnie.) W przypadku kiedy klucz składa się z dwóch części, np. klucza prywatnego $\mathtt{sk}$ i publicznego $\mathtt{pk}$, przyjmiemy, że $\mathsf{KeyGen}(1^n; R) = R = \mathtt{sk}$ a klucz

publiczny pk jest obliczany za pomocą pewnej deterministycznej funkcji f na kluczu prywatnym, tj. $\mathtt{pk} = f(\mathtt{sk})$.

4 Disperse jako Key Derivation Function

W dalszej części pracy będziemy używali następującej notacji. Graf $\mathcal{G}$ będzie M-regularnym grafem dwudzielnym powiązanym z funkcją $\sigma : [N_1] \times [M] \to [N_0]$ następująco: $\mathcal{G}$ składa się z dwóch rozłącznych zbiory wierzchołków $[N_0], [N_1]$ oraz krawędzi pomiędzy $i \in [N_1]$ a $\sigma_{(i,j)} \in [N_0]$ dla każdego $j \in [M]$. Ponadto, graf $\mathcal{G}$ jest disperserem. Od tego momentu przyjmujemy, że $N_0 = N_1 = \ell$.

Na rysunku 2, pokazujemy konstrukcję funkcji Disperse. Dla uproszczenia, identyfikujemy wierzchołki grafu z wartościami w nich zapisanymi. Ilustracja przykładowej funkcji Disperse znajduje się na rysunku 1.

Implementacja $\mathsf{Disperse}_{\mathcal{G}_\sigma}(D, \mathcal{H})$**.**

Wejście: ciąg bitów $D = D_1 \dots D_\ell$ dla $D_1, \dots, D_\ell \in \{0,1\}^n$; $\mathcal{G}_\sigma$, prawy d-regularny graf dwudzielny $(D \sqcup D', E)$, dla którego $D = (D_1, \dots, D_\ell)$ oraz $D' = (D'_1, \dots, D'_\ell)$, takie że $N(D'_i) = \{D_{\sigma_1^i}, \dots, D_{\sigma_d^i}\}$; funkcja $\mathcal{H}\colon \{0,1\}^{dn+\log \ell} \to \{0,1\}^n$
Wyjście: ciąg bitów $D' = D'_1 \dots D'_\ell$ dla $D'_1, \dots, D'_\ell \in \{0,1\}^n$.

Wykonanie:

1. Przypisujemy wartości górnym wartością grafu $\mathcal{G}_\sigma$:
   $D'_i \leftarrow \mathcal{H}\left(i, D_{\sigma_1^i}, \dots, D_{\sigma_d^i}\right)$, dla $i = 1, \dots, \ell$.
2. Zwróć $D' = D'_1, \dots, D'_\ell$.

Rysunek 2: Funkcja Disperse w działaniu

Głównym wynikiem pracy jest poniższe twierdzenie pokazujące, że dla odpowiednio wybranego grafu $\mathcal{G}$ funkcja $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}$ jest bezpieczna i prywatna (dla pewnych rozsądnych, spełnialnych w "świecie rzeczywistym" parametrów).

Twierdzenie 1. *Niech $\mathcal{G}_\sigma$ będzie ℓ^d-regularnym (ℓ^e, η)-prawym disperserem a $\mathcal{H}$ losową wyrocznią. Wtedy dla każdego $\eta < p$ funkcja $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}} : \{0,1\}^{\ell n} \to \{0,1\}^{\ell n}$ jest:*

- $(\Lambda(\lambda) = \lambda + \ell^e(\log q(n) + \log \ell), q(n) = 2^{o(\ell^{1-e})n}, \varepsilon(n) = 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)})$*-prywatna*
- $(\Lambda(\lambda) = \lambda - \ell^e(\log q(n) + n), q(n) = 2^{o(\ell^{1-e})n}, \varepsilon(n) = 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)})$*-bezpieczna.*

Dowód. Dowód twierdzenia znajduje się w rozprawie. □

Kluczowy dla ww. twierdzenia jest lemat pokazujący, że funkcja Disperse jest *jednokierunkowa*. Zauważmy, przytoczona tutaj definicja jednokierunkowości Disperse jest mocniejsza od standardowej definicji funkcji jednokierunkowej. W przypadku tej drugiej, nie możemy powiedzieć, jakie

wiele informacji o argumencie zdradza wartość funkcji, w przypadku Disperse, pokazujemy, że ilość ta jest ściśle ograniczona (z dużym prawdopodobieństwem).

Dla danych $D_1, \ldots, D_\ell$ mówimy, że zapytanie b do losowej wyroczni $\mathcal{H}$ jest *złe* jeśli $b = (i, D_{\sigma_1^i}, D_{\sigma_2^i}, \ldots, D_{\sigma_d^i})$ dla pewnego $i \in \{1, \ldots, \ell\}$. To jest, istnieje taki blok klucza D'_i, że $\mathcal{H}(b) = D'_i$. Poniżej przedstawiamy bardziej szczegółowo co rozumiemy przez jednokierunkowość Disperse:

Lemat 2 (Jednokierunkowość funkcji Disperse)**.** *Niech $\mathcal{G}_\sigma$ będzie ℓ^d-regularnym (ℓ^e, η)-prawym disperserem oraz $D = (D_1, \ldots, D_\ell) \in \{0,1\}^{n\ell}$ zmienną losową o min-entropii $p\ell n$. Wtedy prawdopodobieństwo, że algorytm $\mathcal{A}(\mathsf{Disperse}_{\mathcal{G}_\sigma}(D, \mathcal{H})) \in \mathbf{TM}_{\lambda,q}^{\mathcal{O}(D,\mathcal{H}),\mathcal{H}}$ wykona co najmniej ℓ^e różnych* złych *zapytań wynosi:*

$$\Pr(|\mathbf{indices}_{\mathcal{A}}| \geqslant \ell^e) \leqslant 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)} \ ,$$

dla $\eta \leqslant p$ i $\mathbf{indices}_{\mathcal{A}}$ oznaczające zbiór wszystkich złych zapytań $\mathcal{A}$.

Pokazanie bezpieczeństwa i prywatności funkcji Disperse jest wystarczające by funkcja ta spełniała założenia twierdzenia 1, jednak w rzeczywistym świecie konieczne jest jeszcze pokazanie, że Disperse można efektywnie obliczać.

Twierdzenie 3 (Złożoność obliczeniowa funkcji $\mathsf{Disperse}_{\mathcal{G}_\sigma}(D, \mathcal{H})$)**.** *Niech $\mathcal{H}$ będzie losową wyrocznią, $D \in \{0,1\}^{n\ell}$ oraz $\mathcal{G}_\sigma$ – zdefiniowany wyżej disperser. Przez $K' = \langle K'[i] \rangle_{i=1}^{n\ell}$, gdzie $K'[i] \in \{0,1\}$, oznaczamy klucz otrzymany przez uruchomienie $\mathsf{Disperse}_{\mathcal{G}_\sigma}(D, \mathcal{H})$. Wtedy, dla każdego $i \in 1, \ldots, n\ell$ potrzeba dokładnie*

- *d bloków D by otrzymać wartość $K'[i]$;*
- *$(2d)$ bloków D by otrzymać wartość $(K'[i], \ldots, K'[i+n-1 \pmod{n\ell}])$.*

5 Identyfikacja i podpis cyfrowy dzięki kdf

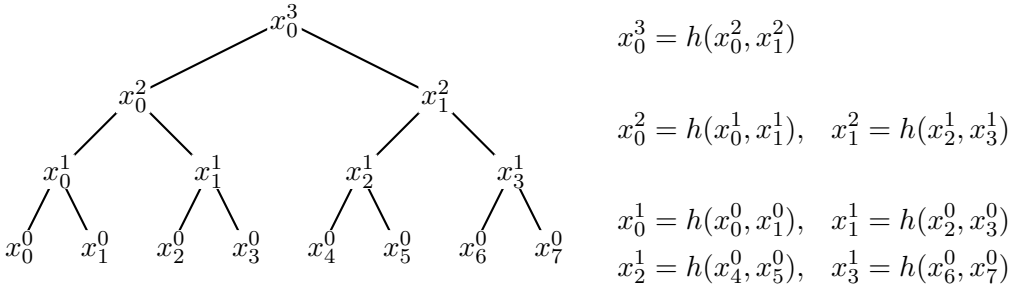
Ważną częścią pracy jest pokazanie, że wprowadzone schematy uzyskiwania klucza kryptograficznego ze zmiennych losowych o niejednostajnym rozkładzie mają zastosowanie w praktyce.

Jako przykład posłużą tu schematy identyfikacji i podpisu cyfrowego. Obydwa działają na tym samym pomyśle wykorzystującym tzw. drzewa Merkla (*Merkle tree*, [32], [31]). Zarówno identyfikacja jak i podpis cyfrowy są jednymi z podstawowych schematów kryptograficznych. Ich szerokie zastosowania sięgają od uwierzytelniania komunikacji pomiędzy bankiem a klientem do zapewnienia bezpieczeństwa transakcji bitcoinowych; bez nich wiarygodna komunikacja za pomocą internetu nie byłaby możliwa.

Drzewa Merkla Drzewa Merkla, są jednym z popularniejszych schematów wykorzystywanych przy identyfikacji. Zapewniają bardzo krótki klucz publiczny, przy jednoczesnym dużym i bezpiecznym kluczu prywatnym. Ponadto, dzięki tej konstrukcji można łatwo i szybko przekształcić dowolną funkcję haszującą $h : \{0,1\}^{2n} \to \{0,1\}^n$ w funkcję $h' : \{0,1\}^{2^k n} \to \{0,1\}^n$ dla dowolnego naturalnego $k > 1$. Definicja drzewa Merkla znajduje się poniżej, rysunek 3 zawiera przykładową konstrukcję drzewa. W rozprawie pokazano, że mogą być one stosowane nawet wtedy, kiedy (a) wartości w liściach nie są jednostajnie losowe, (b) przeciwnik jest wyposażony w wyciek i może poznać część informacji o tajnym kluczu.

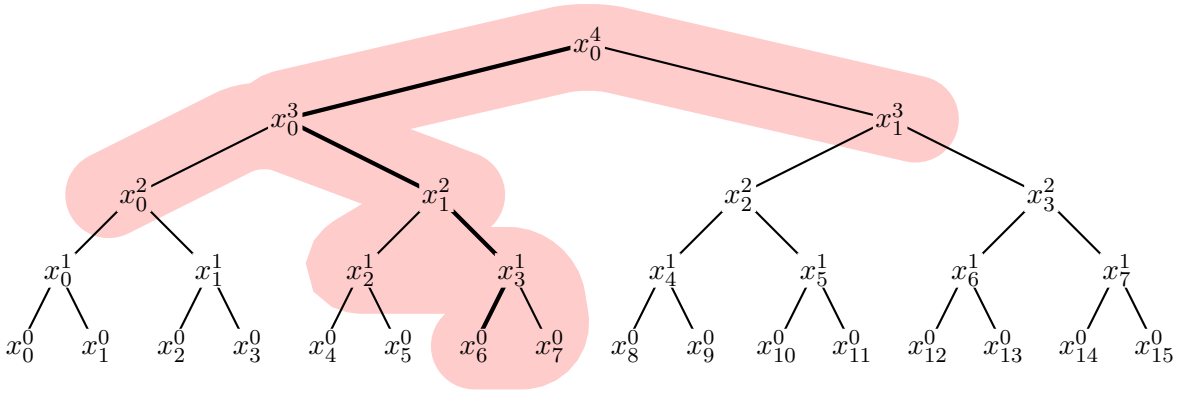
Definicja 4 (Drzewa Merkla dla funkcji h [32])**.** Niech $\mathcal{T}(\ell)$ oznacza pełne drzewo binarne z ℓ liściami i wysokości $\log \ell$, niech x_i^0 (dla $i \in \{0, \ldots, \ell - 1\}$) będzie liściem w $\mathcal{T}(\ell)$ a $x_i^0 \in \{0,1\}^n$ wartością przechowywaną w x_i^0. Dla każdego wierzchołka $x_i^j \in \mathcal{T}(\ell)$, który nie jest liściem oznaczamy przez x_{2i}^{j-1} oraz x_{2i+1}^{j-1} dzieci wierzchołka x_i^j. Takie drzewo $\mathcal{T}(\ell)$ nazywamy *drzewem Merkla* jeżeli dla funkcji haszującej $h : \{0,1\}^{2n} \to \{0,1\}^n$ mamy

$$x_i^j = h\left(x_{2i}^{j-1}, x_{2i+1}^{j-1}\right), \text{ for } j > 0 \text{ and some } x_0^0, x_1^0, \ldots, x_{\ell-1}^0 \ .$$



Rysunek 3: Przykładowe drzewo Merkla

Identyfikacja i podpis cyfrowy w drzewie Merkla Oznaczmy przez $\mathcal{T}$ drzewo Merkla. Ścieżkę identyfikującą od liścia i, nazywamy ciąg elementów drzewa, który zawiera ścieżkę od liścia i do wierzchołka, jak i wszystkich braci wierzchołków znajdujących się na tej ścieżce. Przykładowa ścieżka identyfikująca jest pokazana na rysunku 4. W procesie identyfikacji, weryfikujący $\mathcal{V}$ żąda od dowodzącego $\mathcal{P}$ podania mu pewnej liczby ścieżek identyfikujących wybranych przez $\mathcal{V}$.



Rysunek 4: Przykładowa ścieżka identyfikująca na drzewie Merkla.

W pełnej wersji pracy pokazujemy następujące twierdzenie pokazujące, że drzewa Merkla są użyteczne nawet wtedy, kiedy przeciwnik jest wyposażony w wyciek a klucz potrzebny do identyfikacji nie jest jednostajnie losowy:

Twierdzenie 4 (Bezpieczeństwo identyfikacji)**.** *Dla dowolnego przeciwnika* $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}(K,\mathcal{H}),\mathcal{H}}_{\lambda,q(n)}$ *prawdopodobieństwo, że* $\mathcal{A}$ *wygra w grze opisanej na rysunku 5 dla identyfikacji na drzewie Merkla (rys. 6), dla klucza* K *wybranego jednostajnie losowo z* $\{0,1\}^{n\ell}$, $\zeta(n) < \ell/2k(n)$ *oraz*

Gra bezpieczeństwa dla schematu identyfikacji

Parametry: n – parametr bezpieczeństwa, λ – maksymalna ilość wycieku.

1. **Ustawienia początkowe:** $\mathtt{params} \leftarrow \mathsf{ParamGen}(1^n)$; $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{KeyGen}(1^n; R)$; $\mathcal{A}$ dostaje $\mathtt{pk}, \mathtt{params}$ oraz dostęp do wyroczni $\mathcal{O}_\lambda^{\mathtt{sk}}$.
2. **Faza nauki:** W tej fazie, $\mathcal{A}^{\mathcal{O}_\lambda^{\mathtt{sk}}, \mathcal{P}(\mathtt{sk})}$ ma dostęp black-boksowy do $\mathcal{P}(\mathtt{sk})$, której może zapytać o maksymalnie $\zeta(n)$ wiele dowodów identyfikacji; posiada też dostęp do wyroczni $\mathcal{O}_\lambda^{\mathtt{sk}}$.
3. **Faza mimetyczna:** $\mathcal{A}$ traci dostęp do $\mathcal{P}(\mathtt{sk})$ oraz $\mathcal{O}_\lambda^{\mathtt{sk}}$, przeciwnik $\mathcal{A}$ stara się podszyć pod $\mathcal{P}$ i zidentyfikować przed $\mathcal{V}$.

Rysunek 5: $\mathsf{Auth}_{\zeta(n)}^{\lambda,n}$; Gra bezpieczeństwa schematu identyfikacji

$\lambda \leqslant \frac{1}{c} \cdot (\ell - 2k(n)(\zeta(n) - 1))(n - \log(q(n)r(n)))$, *jest nie większe niż*

$$\left(\frac{1}{c}\right)^{k(n)} + \frac{1}{r(n) - 1} \ ,$$

dla dowolnego $r(n) > 1, c > 1$.

W szczególności, kiedy w twierdzeniu powyżej, jednostajnie losowy klucz K zostanie zastąpiony kluczem otrzymanym z $(\Lambda(\lambda), q(n), \varepsilon(n)$-bezpiecznej funkcji kdf, prawdopodobieństwo, że próba zidentyfikowania się przez przeciwnika $\mathcal{A}$, zakończy się sukcesem jest ograniczone przez

$$\left(\frac{1}{c}\right)^{k(n)} + \frac{1}{r(n) - 1} + \varepsilon(n) \ ,$$

dla dowolnego $r(n) > 1, c > 1$.

5.1 Nieinteraktywny schemat identyfikacji i podpisu cyfrowego oparty na drzewach Merkla

Jak zostało wcześniej powiedziane, nieinteraktywny schemat identyfikacji osiągamy przez zastosowanie heurystyki Fiata-Shamira na interaktywnej wersji schematu. Ponadto, dzięki drobnym modyfikacjom protokołu identyfikacji, otrzymujemy protokół podpisu cyfrowego.

Dokładne opisy nieinteraktywnych schematów identyfikacji i podpisu znajdują się na rys. 6 i 7. Zwracamy tu uwagę, że na ww. rysunkach wprowadzono drobną modyfikację do losowej wyroczni $\mathcal{H}$: przez $\mathcal{H}^{k,\ell}$ oznaczymy losową wyrocznię taką, która zwraca wartość o długości ℓ i wadze Hamminga k, taka wyrocznia może zostać za pomocą dowolnej losowej wyroczni jak to pokazano w [23]. Odpowiedzią na wyzwanie postaci $z = \mathcal{H}^{k,\ell}(r)$ jest wysłanie k ścieżek od liści do korzenia $\mathcal{T}$: $i_1, \ldots, i_k$, gdzie kolejne wartości oznaczają kolejne miejsca wektora binarnego z dla których $z[i] = 1$.

Innym elementem schematu wymagającym wyjaśnienia jest sposób losowania wartości r. W schemacie przestawionym na rysunku 6, wartość r jest losowana do momentu, w którym wszystkie wskazane przez nią ścieżki nie były wcześniej użyte. To wymaganie jest niezbędne dla zachowania bezpieczeństwa schematu. Zauważmy, w wersji interaktywnej identyfikacji też wymagane

Nieinteraktywny protokół identyfikacji oparty na drzewach Merkla

Identyfikujący się $\mathcal{P}$: $\mathcal{T}$ – drzewo Merkla o wartościach $x_1, \ldots, x_\ell$ w liściach.

Przygotowanie:

Identyfikujący się $\mathcal{P}$ oblicza wartość w korzeniu $\mathcal{T}$ i przyjmuje ją jako klucz publiczny $\mathtt{pk}$. Identyfikujący się i weryfikujący ustawiają $\mathbf{S} = \mathbf{R} = \emptyset$ i trzymają swoje niezależne kopie tych zbiorów $\mathbf{S}_\mathcal{V}, \mathbf{R}_\mathcal{V}$ oraz $\mathbf{S}_\mathcal{P}, \mathbf{R}_\mathcal{P}$.

Wykonanie:

1. Identyfikujący się $\mathcal{P}$
   - bierze losową wartość $r \leftarrow \mathbf{Dom}(\mathcal{H}^{k,\ell}) \setminus \mathbf{R}$;
   - ustawia $\mathbf{R} \leftarrow \mathbf{R} \cup \{r\}$;
   - odpytuje wyrocznię losową $\mathcal{H}^{k,\ell}$ na r otrzymując $\mathcal{H}^{k,\ell}(r)$, gdzie $h_0 h_1 \ldots h_{\ell-1}$ jest binarną reprezentacją $\mathcal{H}^{k,\ell}(r)$, przyjmuje $\mathbf{K} = \emptyset$;
   - jeżeli dla każdego i takiego, że $h_i = 1$ zachodzi $i \notin \mathbf{S}_\mathcal{P}$, wtedy $\mathbf{K} \leftarrow \mathbf{K} \cup \{i\}$ oraz $\mathbf{S}_\mathcal{P} \leftarrow \mathbf{S}_\mathcal{P} \cup \{i\}$; w przeciwnym przypadku jeśli istnieje i takie, że $h_i = 1$ oraz $i \in \mathbf{S}_\mathcal{P}$, $\mathcal{P}$ zaczyna od początku wybierając inne r;
   - dla każdego $i \in \mathbf{K}$ oblicza ścieżkę identyfikującą p_i od korzenia aż do i-tego liścia, t.j. wykonuje $\mathsf{p}_i \leftarrow \mathsf{getPath}(\mathcal{T}, \mathtt{pk}, i)$[a];
   - wysyła do $\mathcal{V}$ trójkę $\big(r, \mathcal{H}^{k,\ell}(r), (\mathsf{p}_i)_{i\in\mathbf{K}}\big)$.
2. Na podstawie otrzymanej trójki $\big(r, \mathcal{H}^{k,\ell}(r'), (\mathsf{p}_i)_{i\in\mathbf{K}}\big)$ weryfikujący $\mathcal{V}$ sprawdza następujące:
   - czy $r' \in \mathbf{R}_\mathcal{V}$ lub $\mathcal{H}^{k,\ell}(r') \neq \mathcal{H}^{k,\ell}(r)$ jeżeli tak jest, to $\mathcal{V}$ odpowiada `Reject`.
   - Dla każdego $i \in \mathbf{K}$, weryfikujący $\mathcal{V}$ sprawdza czy:
     - $i \notin \mathbf{S}_\mathcal{V}$, $\mathbf{S}_\mathcal{V} \leftarrow \mathbf{S}_\mathcal{V} \cup \{i\}$;
     - ścieżka jest poprawna przez wywołanie $\mathsf{checkPath}(\mathsf{p}_i, i, \mathtt{pk})$.[b]
3. W tym przypadku, weryfikujący odpowiada `Accept`, w przeciwnym razie odpowiada `Reject`.

[a] Algorytm $\mathsf{getPath}(\mathcal{T}, \mathtt{pk}, i)$ zwraca ścieżkę identyfikującą w drzewie $\mathcal{T}$ od liścia i do korzenia $\mathtt{pk}$. Algorytm jest opisany dokładnie w pełnej wersji pracy.

[b] Algorytm $\mathsf{checkPath}(\mathsf{p}_i, i, \mathtt{pk})$ sprawdza czy podana ścieżka jest poprawną ścieżką od liścia i do wierzchołka drzewa. Algorytm jest opisany dokładnie w pełnej wersji pracy.

Rysunek 6: Przykładowy nieinteraktywny protokół identyfikacji oparty na drzewach Merkla

było by ścieżki wskazane przez weryfikującego $\mathcal{V}$ były nieużyte. Jednocześnie warto podkreślić, że oczekiwana liczba losowań r potrzebnych do uzyskania odpowiedniego argumentu jest ograniczona przez $\frac{2\ell}{\ell - k(n)i}$, gdzie i to numer przeprowadzanej interakcji dla danego drzewa $\mathcal{T}$.

Nieinteraktywny protokół podpisu cyfrowego oparty na drzewach Merkla

Podpisujący $\mathcal{P}$: $\mathcal{T}$ – drzewo Merkla o wartościach $x_1, \ldots, x_\ell$ w liściach.

Przygotowanie:

Podpisujący $\mathcal{P}$ oblicza wartość w korzeniu $\mathcal{T}$ i przyjmuje ją jako klucz publiczny `pk`. Podpisujący i weryfikujący ustawiają $\mathbf{S} = \mathbf{R} = \emptyset$ i trzymają swoje niezależne kopie tych zbiorów $\mathbf{S}_\mathcal{V}, \mathbf{R}_\mathcal{V}$ oraz $\mathbf{S}_\mathcal{P}, \mathbf{R}_\mathcal{P}$.

Wykonanie:

1. Podpisujący $\mathcal{P}$ chcąc podpisać wiadomość m:
   - bierze losową wartość $r \setminus \mathbf{R}$;
   - ustawia $\mathbf{R} \leftarrow \mathbf{R} \cup \{r\}$;
   - odpytuje wyrocznię losową $\mathcal{H}^{k,\ell}$ na parze (m, r) otrzymując $\mathcal{H}^{k,\ell}(m, r)$, gdzie $h_0 h_1 \ldots h_{\ell-1}$ jest binarną reprezentacją $\mathcal{H}^{k,\ell}(m, r)$, przyjmuje $\mathbf{K} = \emptyset$;
   - jeżeli dla każdego i takiego, że $h_i = 1$ zachodzi $i \notin \mathbf{S}_\mathcal{P}$, wtedy $\mathbf{K} \leftarrow \mathbf{K} \cup \{i\}$ oraz $\mathbf{S}_\mathcal{P} \leftarrow \mathbf{S}_\mathcal{P} \cup \{i\}$; w przeciwnym przypadku jeśli istnieje i takie, że $h_i = 1$ oraz $i \in \mathbf{S}_\mathcal{P}$, $\mathcal{P}$ zaczyna od początku wybierając inne r;
   - dla każdego $i \in \mathbf{K}$ oblicza ścieżkę identyfikującą p_i od korzenia aż do i-tego liścia, t.j. wykonuje $\mathsf{p}_i \leftarrow \mathsf{getPath}(\mathcal{T}, \mathtt{pk}, i)$;
   - wysyła do $\mathcal{V}$ trójkę $\big((m, r), \mathcal{H}^{k,\ell}(m, r), (\mathsf{p}_i)_{i \in \mathbf{K}}\big)$.
2. Na podstawie otrzymanej trójki $\big((m, r), \mathcal{H}^{k,\ell}(m', r'), (\mathsf{p}_i)_{i \in \mathbf{K}}\big)$ weryfikujący $\mathcal{V}$ sprawdza następujące:
   - czy $r' \in \mathbf{R}_\mathcal{V}$ lub $\mathcal{H}^{k,\ell}(m', r') \neq \mathcal{H}^{k,\ell}(m, r)$ jeżeli tak jest, to $\mathcal{V}$ odpowiada `Reject`.
   - Dla każdego $i \in \mathbf{K}$, weryfikujący $\mathcal{V}$ sprawdza czy:
     - $i \notin \mathbf{S}_\mathcal{V}$, $\mathbf{S}_\mathcal{V} \leftarrow \mathbf{S}_\mathcal{V} \cup \{i\}$;
     - ścieżka jest poprawna przez wywołanie $\mathsf{checkPath}(\mathsf{p}_i, i, \mathtt{pk})$.
3. W tym przypadku, weryfikujący odpowiada `Accept`, w przeciwnym razie odpowiada `Reject`.

Rysunek 7: Przykładowy podpis cyfrowy oparty na drzewach Merkla

5.2 Ustalanie nowego klucza publicznego

Zastosowanie danych dyskowych do otrzymania klucza `sk` wiąże się z pewną niedogodnością. Klucz zmienia się za każdym razem, kiedy dane ulegają zmianie — każda modyfikacja pliku czy usunięcie go wpływa na wiele bloków klucza. Przez zastosowanie drzewa Merkla do obliczenia klucza publicznego `pk`, klucz `pk` zmienia się ilekroć zmienia się `sk`. (Z własności wyroczni losowej można oczekiwać, że zmiana choć jednego bitu w `sk` pociągnie za sobą zmianę ok. $n/2$ bitów klucza publicznego `pk`).

Dlatego też, potrzebny jest protokół, który pozwoli użytkownikowi na *zalegalizowanie* nowego klucza. Tj. użytkownik dostanie algorytm, który zmienia jego klucz publiczny, kiedy zmieniony został klucz prywatny. W tej części pokazujemy taki protokół. Wskazujemy także, że jest on bardzo wydajny – do uwierzytelnienia nowego klucza publicznego wystarczające jest wysłanie pojedynczej wiadomości.

Obliczanie nowego klucza prywatnego. Jak zostało to pokazane na Rysunku 2, nowy klucz $K' = D'_0, \ldots, D'_{\ell-1}$ jest otrzymywany przez funkcję $\mathsf{Disperse}$, w której kolejne D'_i otrzymywane są następująco:

$$D'_i \leftarrow \mathcal{H}\left(i, D_{\sigma(i,1)}, \ldots, D_{\sigma(i,\ell^d)}\right), \quad \text{dla } i = 0, \ldots, \ell - 1.$$

Zmodyfikujemy tę funkcję w niewielkim stopniu wprowadzając nową zmienną, $\mathtt{ch} \in \{0,1\}^{\log \ell}$, która będzie przechowywała liczbę zmian klucza. Dokładniej, nowa funkcja $\mathsf{Disperse}_{\mathtt{ch}}$ wygląda następująco:

- Oznaczmy przez $D_{\mathtt{ch},i}$ kolejne bloki danych dyskowych $D_{\mathtt{ch}}$,
- Klucz $D'_{\mathtt{ch}}$ jest otrzymywany przez przypisanie do kolejnych bloków $D'_{\mathtt{ch},i}$ wartości

$$D'_{\mathtt{ch},i} \leftarrow \mathcal{H}\left(i + \ell \cdot \mathtt{ch}, D_{\mathtt{ch},\sigma(i,1)}, \ldots, D_{\mathtt{ch},\sigma(i,\ell^d)}\right) \ .$$

Z własności wyroczni losowej wynika, że $D'_{\mathtt{ch}}$ and $D'_{\mathtt{ch}'}$ są niezależnymi zmiennymi losowymi dla $\mathtt{ch} \neq \mathtt{ch}'$.

Legalizowanie nowego klucza publicznego. Legalizacja nowego klucza publicznego $\mathtt{pk}'$ sprowadza się do wysłania podpisanej cyfrowo wiadomości zawierającej $\mathtt{pk}'$. Dokładniej, oznaczmy przez $\mathtt{pk}_{\mathtt{ch}-1}$ klucz publiczny uzyskany ze (starych) danych $D_{\mathtt{ch}-1}$ i przez $\mathtt{pk}_{\mathtt{ch}}$ nowy klucz uzyskany ze zmienionych danych $D_{\mathtt{ch}}$. Użytkownik legalizujący klucz $\mathtt{pk}_{\mathtt{ch}}$ wysyła

$$\left((\mathtt{pk}_{\mathtt{ch}}, r), \mathcal{H}^{k,\ell}(\mathtt{pk}_{\mathtt{ch}}, r), (\mathsf{p}_i)_{i \in \mathbf{K}}\right) \ ,$$

dla losowości r i ścieżek identyfikujących p_i zbudowanych na kluczu z danych $D_{\mathtt{ch}-1}$.

Przechowywanie starych danych. Przedstawiony schemat legalizacji klucza zakłada, że użytkownik może wysłać wiadomość podpisaną za pomocą starego klucza prywatnego. W sytuacji, kiedy klucz jest uzyskiwany z danych dyskowych, oznacza to także, że użytkownik musi przechowywać oprócz nowych danych $D_{\mathtt{ch}}$ także stare dane $D_{\mathtt{ch}-1}$. Potencjalnie, do operacji zmiany klucza użytkownik potrzebuje dodatkowych $n\ell$ bitów przestrzeni dyskowej.

Możemy założyć jednak, że zmiany pomiędzy $D_{\mathtt{ch}-1}$ a $D_{\mathtt{ch}}$ nie są duże. Oznaczmy przez $\mathbf{I}$ zbiór tych bloków danych, które różnią się w $D_{\mathtt{ch}-1}$ i $D_{\mathtt{ch}}$. Użytkownik nie musi przechowywać innych bloków $D_{\mathtt{ch}-1}$ poza tymi wyszczególnionymi w $\mathbf{I}$, zatem jedyna dodatkowa przestrzeń dyskowa, której potrzebuje do zalegalizowania nowego klucza to $|\mathbf{I}| \cdot n$. Przy czym przestrzeń ta może być zwolniona zaraz po legalizacji.

6 Problemy otwarte

Lepsze parametry dla słabszego adwersarza W pracy założono bardzo mocnego adwersarza, który może poznać wyciek nie tylko z klucza prywatnego, ale także z losowej wyroczni. Dokładniej, adwersarz może wybrać pewne y i uzyskać odpowiedź na pytanie typu "podaj mi argument x dla którego zachodzi $\mathcal{H}(x) = y$". Choć wyciek z klucza prywatnego jest dobrze umotywowany praktyką, raczej nie należy oczekiwać by jakikolwiek adwersarz mógł na żądanie odwracać losową wyrocznię. Dlatego też ciekawym problemem badawczym byłoby przerobienie wykorzystanej w rozprawie gry Guessing tak, by ograniczyć możliwości adwersarza do wyciekania tylko z tajnego klucza.

Alternatywne przechodzenie drzewa Merkla W pracach [26, 38] pokazano alternatywne, wydajniejsze obliczeniowo i pamięciowo metody identyfikacji za pomocą drzew Merkla. Ciekawym problemem badawczym zdaje się sprawdzenie, czy algorytmy te pozostają bezpieczne także w modelu z wyciekiem, czy ich większa złożoność nie powoduje, że przeciwnik może zdobyć pewne dodatkowe informacje niezależnie od wycieku.

Usunięcie wyroczni losowej Jak to już zostało wspomniane wcześniej, w implementacjach schematów działających na losowych wyroczniach zastępuje się je funkcjami haszującymi. Jednak takie rozwiązanie sprawia, że dowody bezpieczeństwa schematów przestają być poprawne. Ponadto, zostało pokazane w pracach takich jak [34, 21], że wyrocznia losowa posiada właściwości nieosiągalne w standardowym modelu.

W jednej z ostatnich swoich prac, Lindell [29] wskazał zaproponowaną przez Nielsena w [34] metodę na osłabienie założenia istnienia losowej wyroczni. Zamiast niej, wykorzystuje wyrocznię losową, która nie jest programowalna, tj. jest publicznie dostępna a żaden algorytm nie może wiarygodnie podmienić wartości wynikowej dla zapytania typu $\mathcal{H}(r)$. (Z tej własności korzystaliśmy często pokazując prywatność i bezpieczeństwo funkcji Disperse.)

Korzystając w wyniku Lindella i programowalnego tzw. *common reference string* można sprawdzić, czy zaproponowany w tej pracy schemat pozostaje bezpieczny. Dyskusję na temat modelu z losową wyrocznią można znaleźć w pracy [28].

Odświeżanie sekretu Wykorzystanie drzew Merkla do identyfikacji i podpisów cyfrowych wiąże się nieuniknienie z ograniczoną z góry potencjalną liczbą wykonań. Chociaż to ograniczenie nie jest dużą przeszkodą przez to, że odświeżenie zobowiązania odbywa się poprzez wysłanie pojedynczej wiadomości, wydaje się dobrym pomysłem opracowanie schematu, który nie wymagałby systematycznych odświeżeń zobowiązania.

Identyfikujący się – weryfikujący, wyrównywanie pracy W zaproponowanym rozwiązaniu zdecydowana większość pracy jest wykonywna po stronie identyfikującego się $\mathcal{P}$, ponadto klucz prywatny jest ogromny w porównaniu do klucza publicznego. I to są dobre i praktyczne parametry, co można argumentować faktem, że zwykle trzyma się lokalnie wiele kluczy publicznych różnych użytkowników i niewiele własnych kluczy prywatnych. Z drugiej strony, przepaść pomiędzy rozmiarem klucza prywatnego a publicznego w zaprezentowanym schemacie jest na tyle duża, że pytanie o możliwe skrócenie klucza prywatnego kosztem klucza publicznego wydaje się ciekawym zadaniem badawczym.

Literatura

[1] J. Alwen, Y. Dodis, M. Naor, G. Segev, S. Walfish, and D. Wichs. Public-key encryption in the bounded-retrieval model. In H. Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 113–134. Springer, 2010.

[2] J. Alwen, Y. Dodis, and D. Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 36–54. Springer, 2009.

[3] B. Barak, Y. Dodis, H. Krawczyk, O. Pereira, K. P. F. Standaert, and Y. Yu. Leftover hash lemma, revisited. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.

[4] S. Brain. Youtube company statistics. `http://www.statisticbrain.com/youtube-statistics/`, 2016.

[5] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[6] B. Chess and J. West. *Secure Programming with Static Analysis*. Addison-Wesley Professional, first edition, 2007.

[7] Codenomicon. The heartbleed bug. `http://heartbleed.com`, 2014.

[8] G. Di Crescenzo, R. Lipton, and S. Walfish. Perfectly secure password protocols in the bounded retrieval model. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, pages 225–244, Berlin, Heidelberg, 2006. Springer-Verlag.

[9] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[10] Y. Dodis and A. Smith. Entropic security and the encryption of high entropy messages. In J. Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 556–577. Springer, 2005.

[11] Y. Dodis and Y. Yu. Overcoming weak expectations. In A. Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2013.

[12] K. Durnoga, S. Dziembowski, T. Kazana, M. Zajac, and M. Zdanowicz. Bounded-retrieval model with keys derived from private data. In K. Chen, D. Lin, and M. Yung, editors, *Information Security and Cryptology - 12th International Conference, Inscrypt 2016, Beijing, China, November 4-6, 2016, Revised Selected Papers*, volume 10143 of *Lecture Notes in Computer Science*, pages 273–290. Springer, 2016.

[13] S. Dziembowski. Intrusion-resilience via the bounded-storage model. In S. Halevi and T. Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 2006.

[14] S. Dziembowski, T. Kazana, and D. Wichs. Key-evolution schemes resilient to space-bounded leakage. In P. Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 335–353. Springer, 2011.

[15] S. Dziembowski and K. Pietrzak. Intrusion-resilient secret sharing. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 227–237. IEEE Computer Society, 2007.

[16] S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.

[17] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[18] R. Eveleth. How many photographs of you are out there in the world? `https://www.theatlantic.com/technology/archive/2015/11/how-many-photographs-of-you-are-out-there-in-the-world/413389/`, 2015.

[19] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *J. Cryptology*, 1(2):77–94, 1988.

[20] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. pages 186–194. Springer-Verlag, 1987.

[21] S. Goldwasser and Y. Tauman. On the (in)security of the Fiat-Shamir paradigm. *IACR Cryptology ePrint Archive*, 2003:34, 2003.

[22] V. Guruswami, C. Umans, and S. P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *J. ACM*, 56(4), 2009.

[23] D. Hofheinz, T. Jager, D. Khurana, A. Sahai, B. Waters, and M. Zhandry. How to generate and use universal samplers. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 715–744, 2016.

[24] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc. (N.S)*, 43:439–561, 2006.

[25] M. Howard and D. E. Leblanc. *Writing Secure Code*. Microsoft Press, Redmond, WA, USA, 2nd edition, 2002.

[26] M. Jakobsson, F. T. Leighton, S. Micali, and M. Szydlo. Fractal merkle tree representation and traversal. In M. Joye, editor, *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings*, volume 2612 of *Lecture Notes in Computer Science*, pages 314–326. Springer, 2003.

[27] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[28] N. Koblitz and A. J. Menezes. The random oracle model: a twenty-year retrospective. *Des. Codes Cryptography*, 77(2-3):587–610, 2015.

[29] Y. Lindell. An efficient transform from sigma protocols to NIZK with a CRS and non-programmable random oracle. In Y. Dodis and J. B. Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 93–109. Springer, 2015.

[30] A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer Science+Business Media, Inc., 2005.

[31] R. C. Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, Stanford, CA, USA, 1979. AAI8001972.

[32] R. C. Merkle. A digital signature based on a conventional encryption function. In C. Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer Berlin Heidelberg, 1988.

[33] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. *SIAM J. Comput.*, 41(4):772–814, 2012.

[34] J. B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2002.

[35] J. Radhakrishnan and A. Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *Siam Journal on Discrete Mathematics*, 13:2000, 2000.

[36] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, Feb. 1978.

[37] B. Schneier. Heartbleed. `https://www.schneier.com/blog/archives/2014/04/heartbleed.html`, 2014.

[38] M. Szydlo. Merkle tree traversal in log space and time. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 541–554. Springer, 2004.