# V. PRAKTYCZNE ZASTOSOWANIA KRYPTOLOGII

## PROGRAMY JEDNORAZOWE: KRÓTKIE WPROWADZENIE

**Tomasz Kazana**

Instytut Informatyki, Wydział Matematyki, Informatyki i Mechaniki
Uniwersytet Warszawski

**Streszczenie.** Niniejszy dokument stanowi skróconą wersję pracy „One-time Programs with Limited Memory" autorstwa Konrada Durnogi, Stefana Dziembowskiego, Tomasza Kazany oraz Michała Zająca, prezentowanej na konferencji INSCRYPT 2013 [11].
Praca bada pojęcie *programów jednorazowych* wprowadzonych na konferencji CRYPTO'08 przez Shafi Goldwasser et al. Program jednorazowy to urządzenie zawierające program $C$ oraz posiadające własność, iż może być on wykonany tylko raz, na wybranym wejściu. Goldwasser et al. pokazali jak zaimplementować programy jednorazowe, używając specyficznych rozwiązań sprzętowych, tzw. OTM-ów. (ang. One-Time Memory).
Ta praca podaje inną implementację programów jednorazowych, działającą w tzw. modelu obliczeń SBA. Charakterystyczne cechy tego modelu to ograniczona pamięć, wycieki oraz użycie losowej wyroczni.

**Słowa kluczowe:** funkcje pseudolosowe; urządzenia jednorazowe; programy jednorazowe; zaciemnianie obwodów.

## 1. Wprowadzenie

Pojęcie jednorazowych programów zostało wprowadzone przez Goldwasser et al [20]. Nieformalnie mówiąc, program jednorazowy to urządzenie $D$ zawierające program w $C$, który posiada następujące własność: program $C$ może być wykonany na co najwyżej jednym wejściu. Innymi słowy, każdy użytkownik, nawet złośliwy, który uzyskuje dostęp do $D$, powinien być w stanie nauczyć się wartości $C(x)$ dla dokładnie jednego wybranego przez siebie $x$. Jak twierdzi Goldwasser et al, programy jednorazowe mają ogromny potencjał zastosowań w ochronie oprogramowania, tokenów elektronicznych i elektronicznych środków pieniężnych.

Zwróćmy uwagę na następującą obserwację: bezpieczeństwo programów jednorazowych nie może być oparte tylko na oprogramowaniu. Innymi słowy, musi zawsze zawierać pewne założenia o właściwościach fizycznych urządzenia D. Istotnie, jeśli zakłada się, że cała zawartość $C$ w $D$ może być swobodnie czytana, to przeciwnik może tworzyć swoje własne kopie

$D$ i obliczyć $C$ tyle razy, ile chce. Stąd naturalne pytanie jakie „własności fizyczne" są potrzebne do stworzenia programów jednorazowych. Oczywiście, trywialny sposób to po prostu założyć, że $D$ jest w pełni zaufany, czyli przeciwnik nie może odczytać lub zmodyfikować jego zawartość. Oczywiście wówczas, można po prostu umieścić dowolny program $C$ na $D$, dodając dodatkowe instrukcje, aby umożliwić tylko jedno wykonanie $C$. Niestety, okazuje się, że takie założenie jest często nierealne. Liczne prace nad tzw. wyciekami (ang. leakage) i wirusami pokazują, że w praktyce stworzenie urządzenia odpornego na wycieki i wirusy jest trudne, jeśli nie niemożliwe. Dlatego pożądane jest, aby oprzeć programy jednorazowe na słabszych założeniach fizycznych.

Konstrukcja Goldwasser et al. opiera się na następującym założeniu fizycznym: $D$ jest wyposażony w specjalne gadżety, tzw. OTM (ang. One-Time Memory). Przed umieszczeniem OTM w $D$, gadżet OTM może zostać zainicjalizowany z parą wartości $(K_0, K_1)$.Program $C$, który jest przechowywany na $D$ może później poprosić OTM o dokładnie jedną z wartości $K_i$. Główną cechą bezpieczeństwa OTM jest to, że w żadnym wypadku nie jest możliwe jednoczesne poznanie zarówno $K_0$ jak i $K_1$. Technicznie, może to być zrealizowane przez (a) przechowywanie na każdym OTM flagi $u$ początkowo ustawionej na 0, która zmienia swą wartość na 1 po pierwszym zapytaniu do tego OTM, oraz (b) dodanie wymogu, że jeśli $u = 1$, to OTM nie odpowiada na żadne zapytania. Okazuje się, że przy założeniu o posiadaniu OTM w $D$, można skonstruować ogólny kompilator, który przekształca dowolny program $C$ (w postaci układu logicznego) do jednorazowego programu. To jest właśnie główny wynik pracy Goldwasser et al [20]: zamiast nierealistycznych założeń o całym $D$, zakłada ona tylko istnienie bezpiecznych gadżetów OTM. Bezpiecznych, tzn. odpornych na wycieki i manipulacje.

W naszej pracy staramy podejść do problemu inaczej. Nie zakładamy istnienia żadnych dodatkowych bezpiecznych gadżetów. W zamian za to, przyjmujemy, że urządzenie ma ograniczoną pamieć wewnętrzną oraz (aktywny) przeciwnik ma pewne ograniczenie na rozmiar wycieku. Te założenia formalnie opisuje model SBA.

W pełnej wersji pracy [11] pokazujemy dokładną konstrukcję i dowód na istnienie programów jednorazowych w modelu SBA.

# 2. SBA model

## 2.1. Wstęp

SBA model – wprowadzony w pracy [15] – dotyczy problemów kryptograficznych w środowisku z wyciekami informacji oraz aktywnym przeciwnikiem. Zaczniemy od wyjaśnień intuicyjnych.

Rozważamy ogólny schemat: istnieje długi ciąg bitów $R$ (w zamiarze tajny), z którego potrafimy coś obliczyć (powiedzmy $f(R)$), a chcemy aby pozostało to sekretem. W trakcie eksperymentu pojawi się przeciwnik, który *czegoś* o $R$ się dowie, ale prawie zawsze okaże się ta wiedza *zbyt mała*, aby wnioskować coś na temat $f(R)$.

We współczesnej kryptografii istnieje trend konstruowania protokołów odpornych na podobnych przeciwników. Zwykle albo zakładamy iż przeciwnik jest pasywny, co oznacza, że nie wpływa na $R$ w trakcie wykonania obliczeń przez uczciwego użytkownika, a jedynie wybiera funkcję $g$ i poznaje $g(R)$[1]. Oczywiście $g$ nie może być dowolne, bo wówczas gdy $g = f$, to przeciwnik poznaje cały sekret od razu. Ta uwaga sugeruje, że rozsądne jest przyjęcie założenia, że $g$ musi być wybrane z jakiejś (możliwie szerokiej) klasy, do której nie należy $f$. Przykładem jest założenie, że zbiór wartości $g$ jest istotnie mniejszy niż $f$, tzn. $|g(R)| \ll |f(R)|$. Przykłady prac o atakach pasywnych: [1, 4, 5, 8–10, 17–19, 21–26]. Inne założenie to tak zwany przeciwnik aktywny[2], który może złośliwie podmieniać $R$ na wybrane $R'$ czy wręcz zmieniać algorytm liczenia funkcji z $f$ na wybrane $f'$. Przykłady prac z tej dziedziny: [2, 3, 6, 7, 12, 13, 16].

W SBA[3] modelu podjęta jest próba połączenia tych dwóch paradygmatów. To znaczy opisany niżej model zakłada, że istnieje aktywny wirus ($\mathcal{A}_{small}$), który dodatkowo może spowodować wybrany *wyciek*. Przy pewnych ograniczeniach pokazujemy, że skonstruowane schematy wciąż pozostają bezpieczne.

## 2.2. Motywacja dla SBA-modelu

Model SBA próbuje wypełnić lukę między światem praktyków i teoretyków. Z jednej strony bezpieczeństwo jest w pełni udowodnione, ale jak zwykle w kryptografii, przyjmuje się przy tym pewne założenia postulowane przez praktyków, którzy *wierzą*, że pewne konstrukcje są bezpieczne.

[1] Popularnie mówiąc: przeciwnik powoduje wyciek $g$.

[2] Popularnie mówiąc: przeciwnik jest wirusem.

[3] Skrót SBA pochodzi od *Small and Big Adversary*.

Konkretnie, korzystamy z założenia o istnieniu losowej wyroczni, próbującym uchwycić ideę funkcji haszujących.

### 2.3. Losowa wyrocznia

Losowa wyrocznia (ang. *random oracle*) to program, który na dowolne zapytanie odpowiada losowo (a więc prawdopodobieństwo wyniku obliczeń jest jednostajnie rozłożone na przeciwdziedzinie), chyba że dane zapytanie pojawiło się już wcześniej. Wówczas losowa wyrocznia odpowiada tak samo, jak wcześniej.

### 2.4. Formalna definicja SBA–modelu

Przez *przeciwnika* będziemy rozumieć parę algorytmów $\mathcal{A} = (\mathcal{A}_{small}, \mathcal{A}_{big})$, które uruchamiane są jednocześnie oraz mogą się komunikować. Oba algorytmy mają dostęp do wspólnej losowej wyroczni $H$. Zakładamy, iż tylko $\mathcal{A}_{small}$ ma bezpośredni dostęp do tajnego ciągu bitów $R$. Wynikiem obliczeń przeciwnika jest wynik obliczeń $\mathcal{A}_{big}$. A więc celem przeciwnika jest, aby algorytm $\mathcal{A}_{big}$ obliczył jakiś sekret zależny od $R$.[4]

Bedziemy oznaczać $\mathcal{A}^{H(\cdot)}(R) = \left(\mathcal{A}_{big}^{H(\cdot)} \leftrightarrows \mathcal{A}_{small}^{H(\cdot)}(R)\right)$ jednoczesne wykonanie $\mathcal{A}_{big}$ oraz $\mathcal{A}_{small}$, gdzie $\mathcal{A}_{small}$ na wejściu dostaje $R$ i oba algorytmy mają dostęp do losowej wyroczni $H(\cdot)$. Jak wspomniano wyżej, wyjście tak opisanego $\mathcal{A}$ jest definiowane jako wyjście samego $\mathcal{A}_{big}$.

W większości twierdzeń będziemy twierdzić, że $\mathcal{A}$ nie jest w stanie *czegoś* policzyć, o ile spełnione są następujące założenia (dla konkretnych $s$, $c$ oraz $q$ podawanych w twierdzeniach):

- $\mathcal{A}_{small}$ ma ograniczoną pamięć przez $s$.
- Komunikacja od $\mathcal{A}_{small}$ do $\mathcal{A}_{big}$ jest ograniczona przez $c$. [5] W drugą stronę jest nieograniczona.
- Liczba pytań jakie $\mathcal{A}_{small}$ i $\mathcal{A}_{big}$ mogą łącznie zadać losowej wyroczni jest ograniczona przez $q$.

[4] Zwykle $\mathcal{A}_{small}$ może łatwo obliczyć sekret ponieważ ma dostęp do $R$. Wracając do intuicji: należy myśleć, że $\mathcal{A}_{small}$ to mały wirus zainstalowany na urządzeniu zawierającym $R$, a dopiero $\mathcal{A}_{big}$ to *prawdziwy* przeciwnik, który chce poznać sekret.

[5] Intuicyjnie założenia dotyczące $\mathcal{A}_{small}$ wydają się rozsądne, gdyż $\mathcal{A}_{small}$ to wirus, a ten jest ograniczony przez zewnętrzne urządzenie na którym jest zainstalowany. Innymi słowy, aby je spełnić, wystarczy odpowiednio przygotować urządzenie, na którym przechowywany jest $R$.

W wyżej wymionym przypadku będziemy pisać, że $\mathcal{A}$ jest $(s, c, q)$ ograniczony.

Czasem, poza $R$, przeciwnik $\mathcal{A}$ może mieć dodatkowe wejście $x$. Wówczas zakładamy, że dane $x$ początkowo znajduje się na wejściu $\mathcal{A}_{big}$.

## 3. Programy jednorazowe (OTP)

**Idea** Tajny ciąg $R$ zawiera opis pewnego programu $C$, który może zostać wykonany tylko raz, dla wybranego wejścia. Innymi słowy, dowolny użytkownik (również złośliwy) dostaje urządzenie z programem, ale nie wie, co to za program. Pokazujemy, że jedyne czego się dowie to wartość $C(x)$ dla dokładnie jednego $x$.

**Definicja N**iech $C : \{0,1\}^n \to \{0,1\}^m$ będzie programem (rozumianym jako obwód logiczny). Ciąg bitów $R_C$ to pewien ciąg bitów, generowany z $C$ przez efektywny niedeterministyczny algorytm używający $H$. Algorytm $D$ jest $(c, s, \epsilon)$–one-time programem dla klasy wszystkich funkcji $\{0,1\}^n \to \{0,1\}^m$, gdy:

- Algorytm $D$ dla danego $R_C$ oraz $x \in \{0,1\}^n$ oblicza $C(x)$, nawet gdy jest $(s, c, q)$ ograniczony.
- Istnieje symulator $S$ z dostępem do wyroczni jednokrotnego dostępu obliczającej $C$ (ale bez dostępu do $R_C$) taki, że dla dowolnego przeciwnika $\mathcal{A}$ mającego dostęp do $R_C$ i $(s, c, q)$–ograniczonego nie da się odróżnić wyniku obliczeń $S$ od wyniku obliczeń $\mathcal{A}$ z prawdopodobieństwem większym niż $\epsilon$.[6]

**Wynik** Dla dowolnych $(n, m)$ istnieje $(c, s, \epsilon)$–program jednorazowy dla klasy wszystkich funkcji $\{0,1\}^n \to \{0,1\}^m$ w SBA–modelu z paramterami $c, s, \epsilon$ opisanymi w pełnej wersji pracy [11].

**Wynik na tle dziedziny** Pojęcie *One-time program* zostało wprowadzone przez Goldwasser et al. w [20]. Autorzy dowodzą tam istnienia OTP w modelu z założeniami o tzw. OTM (one-time memory), szczegóły w [20].

**Konstrukcja** Szczegóły konstrukcji można znaleźć w pełnej wersji pracy [11]. Tutaj przedstawimy tylko pewne intuicje i „smak" idei.

Głównym pomysłem technicznym jest próba symulowania rozwiązania zaproponowanego przez Goldwasser et al., ale bez OTM-ów używanych

[6] Bardziej precyzyjnie: nie istnieje żaden algorytm (*odróżniacz*), który odróżnia wyżej opisane wyniki obliczeń z prawdopodobieństwem większym niż $\frac{1}{2} + \epsilon$, jeśli liczba jego pytań do wyroczni jest ograniczona przez $q$.

w oryginalnej konstrukcji. Zamiast OTM-ów używamy – w dość zaskakujący i nietrywialny sposób – idei losowych funkcji jednorazowych, wziętych z [15].

Dzięki ograniczeniom pamięci wewnętrznej urządzenia, pokazujemy, że nie jest możliwe obliczenie losowej funkcji jednorazowej $f$ na więcej niż jednym wejściu. To już pozwala nam na symulowanie OTM-ów, gdyż jako zawartość pamięci OTMa, przyjmujemy $f(0)$ oraz $f(1)$. Wartości te są losowe i nie mamy nad nimi kontroli. Nie jest to jednak problem, gdyż w konstrukcji Goldwasser et al. klucze trzymane w OTM-ach też są losowe.

## Literatura


[1] A. Akavia, S. Goldwasser, and V. Vaikuntanathan, *Simultaneous hardcore bits and cryptography against memory attacks*, In TCC, 2009.

[2] J. Alwen, Y. Dodis, M. Naor, G. Segev, S. Walfish, and D. Wichs, *Public-key encryption in the bounded-retrieval model*, In EUROCRYPT, 2010.

[3] J. Alwen, Y. Dodis, and D. Wichs, *Leakage-resilient public-key cryptography in the boundedretrieval model*, In CRYPTO, 2009.

[4] Z. Brakerski and S. Goldwasser, *Circular and leakage resilient public-key encryption under subgroup indistinguishability (or: Quadratic residuosity strikes back)*, CRYPTO, 2010.

[5] Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikuntanathan, *Cryptography resilient to continual memory leakage*, FOCS, 2010.

[6] D. Cash, Y. Z. Ding, Y. Dodis, W. Lee, R. J. Lipton, and S. Walfish, *Intrusion-resilient key exchange in the bounded retrieval model*, In TCC, 2007.

[7] G. D. Crescenzo, R. J. Lipton, and S. Walfish, *Perfectly secure password protocols in the bounded retrieval model*, In TCC, 2006.

[8] F. Davi, S. Dziembowski, and D. Venturi, *Leakage-resilient storage*, SCN, 2010.

[9] Y. Dodis, S. Goldwasser, Y. T. Kalai, C. Peikert, and V. Vaikuntanathan, *Public-key encryption schemes with auxiliary inputs*, In TCC, 2010.

[10] Y. Dodis, K. Haralambiev, A. Lopez-Alt, and D. Wichs, *Cryptography against continuous memory attacks*, FOCS, 2010.

[11] K. Durnoga, S. Dziembowski, T. Kazana, and M. Zajac, *One-time programs with limited memory*, In INSCRYPT, 2013.

[12] S. Dziembowski, *Intrusion-resilience via the bounded-storage model*, In TCC, 2006.
[13] S. Dziembowski, *On forward-secure storage*, In CRYPTO, 2006.
[14] S. Dziembowski, T. Kazana, and D. Wichs, *Key-evolution schemes resilient to space-bounded leakage*, In CRYPTO, pages 335.353, 2011.
[15] S. Dziembowski, T. Kazana, and D. Wichs, *One-time computable self-erasing functions*, In TCC, pages 125.143, 2011.
[16] S. Dziembowski and K. Pietrzak, *Intrusion-resilient secret sharing*, In FOCS, pages 227.237, 2007.
[17] S. Dziembowski and K. Pietrzak, *Leakage-resilient cryptography*, In FOCS, 2008.
[18] ECRYPT. The Side Channel Cryptanalysis Lounge http://www.crypto.rub.de/en_sclounge.html.
[19] S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum, *Leakage-resilient signatures*, In TCC, 2010.
[20] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, *One-time programs* In D. Wagner, editor, CRYPTO, volume 5157 of LNCS, pages 39.56, 2008.
[21] Y. Ishai, A. Sahai, and D.Wagner, *Private Circuits: Securing Hardware against Probing Attacks*, In CRYPTO, 2003.
[22] J. Katz and V. Vaikuntanathan, *Signature schemes with bounded leakage resilience*, In ASIACRYPT, pages 703.720, 2009.
[23] S. Micali and L. Reyzin, *Physically observable cryptography (extended abstract)*, In TCC, 2004.
[24] M. Naor and G. Segev, *Public-key cryptosystems resilient to key leakage*, In Advances in Cryptology - CRYPTO, August 2009.
[25] K. Pietrzak, *A leakage-resilient mode of operation*,In EUROCRYPT, 2009.
[26] F.-X. Standaert, T. Malkin, and M. Yung, *A unified framework for the analysis of side-channel key recovery attacks*, In EUROCRYPT, 2009.

## ONE-TIME PROGRAMS WITH LIMITED MEMORY

**Abstract.** We reinvestigate a notion of one-time programs introduced in the CRYPTO 2008 paper by Goldwasser et al. A one-time program is a device containing a program C, with the property that the program C can be executed on at most one input. Goldwasser et al. show how to implement one-time programs on devices equipped with special hardware gadgets called one-time memory tokens.

We provide an alternative construction that does not rely on the hardware gadgets. Instead, it is based on the following assumptions: (1) the total amount of data that can leak from the device is bounded, and (2) the total memory on the device (available both to the honest user and to the attacker) is also restricted, which is essentially the model used recently by Dziembowski et al. (TCC 2011, CRYPTO 2011) to construct one-time computable pseudorandom functions and key-evolution schemes.

**Keywords:** pseudorandom functions; one-time device; one-time program; circuit garbling.