Název projektu: Nástroje pro verifikaci bezpečnosti kryptografických zařízení s využitím AI

Identifikační kód: VJ02010010

Odborná zpráva - Etapa 4

Období: 07/2022 - 12/2022

Příjemce:	Vysoké učení technické v Brně Antonínská 548/1 Brno 60190
Jméno hlavního řešitele:	doc. Ing. Jan Hajný, Ph.D. Tel.: +420541146961 Email: hajny@vut.cz
Další účastník 1:	Masarykova univerzita Žerotínovo náměstí 617/9 Brno 602 00
Jméno řešitele:	doc. RNDr. Petr Švenda, Ph.D. Tel.: +420549491878 Email: svenda@fi.muni.cz
Další účastník 2:	České vysoké učení technické v Praze Jugoslávských partyzánů 1580/3 Praha 160 00
Jméno řešitele:	Ing. Martin Novotný, Ph.D. Tel.: +420224358715 Email: martin.novotny@fit.cvut.cz

Obsah

Číslo a název aktivity: Etapa 4	
Období řešení:	07/2022 - 12/2022
Cíl aktivity:	Analýza a volba kryptografických primitiv pro SW implementaci, návrh programového vybavení pro analýzu implementací kryptografických algoritmů pro SW čipových karet.
Krátký popis řešení:	Volba a analýza vhodných nástrojů a kryptografických primitiv s aplikačním garantem. Identifikace scénářů využití. Návrh, implementace a vyhodnocení první iterace nástrojů.
Řešitel zodpovědný za realizaci:	Petr Švenda (Masarykova univerzita).
Stav plnění aktivity:	Splněno
Výstupy:	1x shrnující zpráva (CZ), 1x přílohy popisující provedení nástroje JCProlerNext (mgr práce, EN), 6x GitHub repositář nástrojů a přehledové databáze. Nová nebo rozšířená implementace pěti analytických nástrojů, přehledová databáze existujících nástrojů pro testovaní časové konstantnosti implementací.
Výsledky:	D - stať ve sborníku: SVENDA, Petr, Rudolf KVASNOVSKY, Imrich NAGY a Antonin DUFKA. JCAlgTest: Robust Identification Metadata for Certified Smartcards. Proceedings of the 19th International Conference on Security and Cryptography. SCITEPRESS - Science and Technology Publications, 2022, 2022, 597-604. ISBN 978-989-758-590-6. Dostupné z: doi:10.5220/0011294000003283 ISTA: VJ02010010-V8
Doložení splnění aktivity:	Odborná zpráva – Etapa 4
Shrnutí:	Všechny podmínky pro přechod do další aktivity byly splněny.

1 Úvod

Veškeré plánované činnosti v rámci Etapy 4 byly úspěšně dokončeny a proběhla prezentace výsledků aplikačnímu garantovi v rámci čtyř osobních schůzek a workshopů (červenec, září a 2x listopad 2022).
V rámci etapy #4 (07/2022 - 21/2022) proběhlo mapování oblastí zájmu aplikačního garanta v doméně bezpečné implementace kryptografických primitiv a programového vybavení pro analýzu bezpečnosti implementací kryptografických algoritmů pro čipové karty a softwarové knihovny. Na základě diskuze byly vytypovány scénáře použití, cílové platformy a funkčnost požadovaných nástrojů pokrývající jak existující nástroje vyžadující rozšíření a úpravy, tak nové nástroje pro dosud nepokryté oblasti. Následně proběhl návrh a první iterace implementací s průběžnou demonstrací garantovi a zapracování požadovaných úprav.
Tato zpráva pokrývá princip fungování, základní použití a vyhodnocení dosavadních výsledků pro 5 analytických nástrojů v různé fázi vývoje a jednu přehledovou rešerši zaměřenou na existující nástroje pro kontrolu časové konstantnosti implementací.

Průběžné výsledky byly prezentovány aplikačnímu garantovi (NUKIB) v těchto termínech:

- **1.7.2022, Praha.** Mapování oblastí zájmu aplikačního garanta, představení možných analytických nástrojů pro další rozvoj.
- **8.9.2022, Praha.** Diskuze ohledně nástrojů pro statistickou analýzu náhodnosti.
- **30.10.2022 (dopoledne), Praha.** Prezentace dosavadních výstupů projektu.
- **30.10.2022 (odpoledne), Praha.** Rozšířená diskuze scénářů užití a cílových platforem.

2 Analýza a volba kryptografických primitiv pro softwarovou implementaci, návrh programového vybavení pro analýzu implementací kryptografických algoritmů pro software čipových karet

V rámci této sekce poskytujeme popis nástrojů a databází se zaměřením na bezpečnost implementací vyvíjených nebo rozvíjených v rámci prací na projektu v letošním roce. Uvedený popis pokrývá základní principy fungování nástrojů a ukázky produkovaných výstupů. Pro detailnější informace odkazujeme na příslušnou dokumentaci, reporty a publikované články.
Pokrýváme následující nástroje:

- **ct-tools**: databáze a analýza existujících nástrojů pro detekci časové nekonstantnosti implementací. Databáze poskytuje přehled nástrojů dle typu programovacího jazyku, použité platformy, způsobu testovaní a poskytovaných garancí výsledku (sekce 2.1).
- **JCProfilerNext**: analýza rychlosti běhu, paměťové náročnosti a používaných balíků JavaCard aplikací. Analýza probíhá probíhá přímo na cílové kartě, umožňuje detekovat části časově náročné kódu (rychlostní optimalizace) a detekci časově nekonstantních částí kódu (sekce 2.2).
- **JCMathLib**: matematická knihovna pro platformu JavaCard poskytující nízkoúrovňové objekty typu velká čísla (Bignat) nebo bod na křivce (ECPoint) a jejich související operace, které nejsou dostupné ve standardním JavaCard API (sekce 2.3).
- **ec-detector**: detekce použitých typů eliptických křivek v existujícím zdrojovém kódu, využitelné pro iniciální audit kryptografického kódu (sekce 2.4).
- **ECTester**: rozsáhlá otevřená sada testů korektnosti implementace aplikací využívajících kryptografii na bázi eliptických křivek. Umožňuje testovat implementace na čipových kartách i softwarové kryptografické knihovny (sekce 2.5).
- **JCAlgTest**: aplikace pro sběr a analýzu funkčních i výkonnostních metadat získaných z čipových karet a související otevřená databáze (sekce 2.6).

V rámci projektu jsou nástroje vyvíjeny či upravovány dle potřeb aplikačního garanta a průběžně konzultovány na osobních setkáních.

2.1 *ct-tools*: databáze nástrojů pro detekci časové nekonstantnosti kryptografických implementací

Útoky časovým postranním kanálem (timing attacks) představují vážnou hrozbu pro kryptografické implementace. Využívají závislosti délky běhu kódu na tajné informaci (typicky kryptografický klíč) použité během provádění této operace. Mezi časové útoky řadíme také útoky na vyrovnávací paměť (cache) nebo mikroarchitektonické útoky, které jsou schopny získat informace o adresách paměti, ke kterým jejich cíl přistupuje, nebo informace o adresách instrukcí prováděných jejich cílem. Pokud posloupnost těchto adres závisí na kryptografickém klíči nebo jiné tajné informaci, může je útočník získat pomocí časových či cache útoků. Základní technikou obrany proti časovým útokům je programování kódu s konstantním časem, kdy jsou všechny

přístupy do paměti nebo podmíněné větvení v programu prováděny bez závislosti na tajných informacích.
Technika programování kódu s konstantním časem je dobrým základem pro obranu proti časovým útokům. Jelikož se jedná o programovací techniku, pravděpodobnost lidské chyby je nezanedbatelná, a proto je nutné testovat a ověřovat odolnost kryptografického softwaru proti časovým útokům, nejlépe pomocí automatizovaných nástrojů. Přestože těchto nástrojů existuje větší množství, princip jejich fungování, cílový programovací jazyk anebo platforma, aktivita vývoje a další omezení neumožňuje snadný výběr vhodného nástroje. Proto jsme provedli rešerši dostupných nástrojů, porovnali jejich vlastnosti a identifikovali malou podmnožinu nástrojů vhodných pro další analýzu.

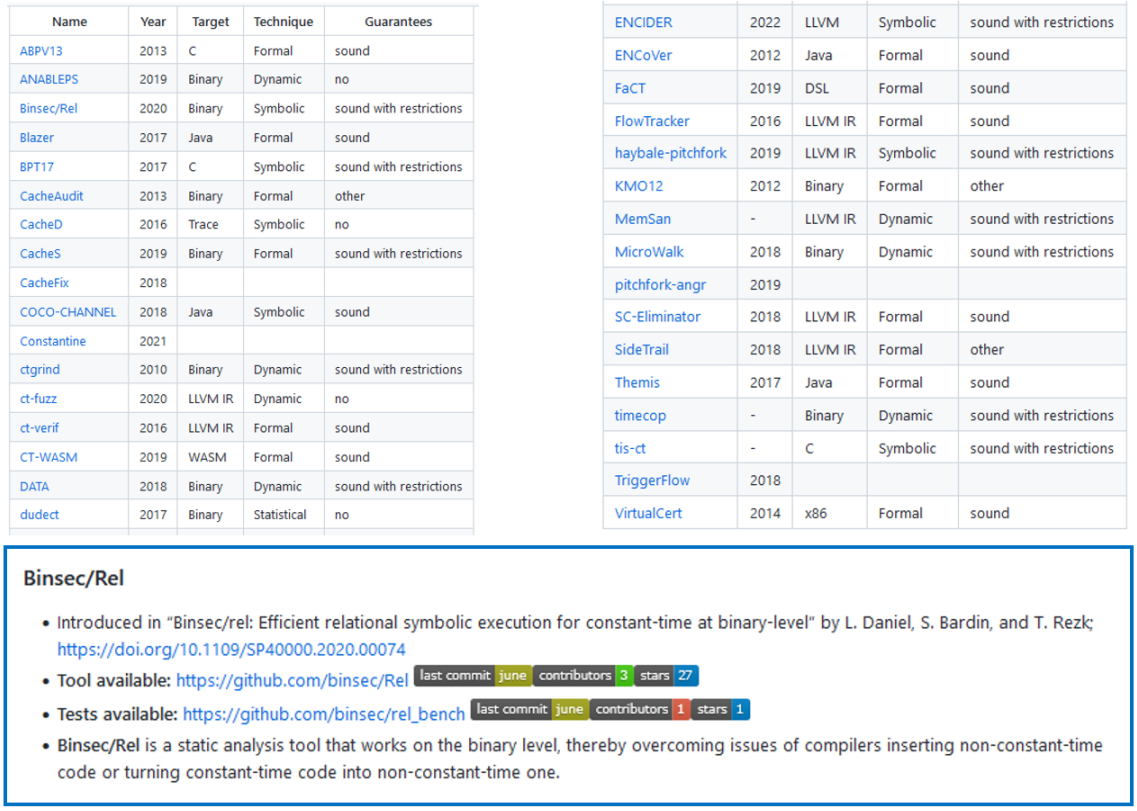
2.1.1 Kritéria hodnocení

Představené nástroje jsou vyhodnoceny a rozděleny do kategorií na základě několika charakteristik. Prvním a nejvýznamnějším kritériem je *obecný přístup* použitý nástrojem z hlediska požadavků na spuštění kódu. Nástroje dělíme na dynamické, které cílový kód spouští a statické, které jej nespouští. Dynamické nástroje obvykle nějakým způsobem instrumentují cílový program, pozorují jeho běh s různými vstupy a poté vyhodnocují, zda rozdíly v běhu unikají pozorovatelným postranním časovým kanálem. Statické nástroje naproti tomu obvykle používají techniky ze statické analýzy a formální verifikace programů k analýze kódu k určení, zda je kód časově konstantní s ohledem na daný model úniku (tajné) proměnné.
Druhým významným kritériem, které odlišuje mnoho nástrojů, je *úroveň vstupu*, na kterém pracují. Řada nástrojů pracuje na úrovni zdrojového kódu, většina s jazykem C, některé s vlastním doménově specifickým jazykem. Druhou kategorií jsou nástroje volící práci na nižší úrovni, často v tzv. LLVM IR (Low Level Virtual Machine Intermediate Representation), jazyku podobnému assembleru ve formě SSA (Static Single Assignment), který se používá v skupině nástrojů LLVM. Protože všechny uvedené vstupní úrovně jsou nad úrovní assembleru/binárního kódu, jsou vystaveny riziku, že zranitelnost vůči časovému útoku bude zavedena až později kompilátorem a nebude nástrojem pracujícím na této úrovni zachycena. Toto riziko je reálné, protože kompilátory obecně nemají za cíl udržovat výstup bez postranních kanálů a typicky nevědí, které proměnné obsahují tajnou citlivou hodnotu. Nástroje pracující přímo s kompilovanými binárními soubory tak obecně poskytují nejvyšší záruky odolnosti vůči časovým útokům.
Další podstatnou vlastností nástrojů je použitý *model úniku* a možnost jeho konfigurace. Za model úniku považujeme to, co nástroj považuje za postranní kanály sledovatelné útočníkem. Existují tři běžné modely úniku, které se v nástrojích často kombinují. *Model úniku větvením* považuje za problém všechny instrukce větvení (změny čítače programu) podmíněné tajnými hodnotami. *Model úniku přístupem do paměti* považuje za problém všechny přístupy do paměti, jejichž adresa závisí na tajných hodnotách. *Model úniku operandem* považuje za problém všechna použití konkrétních instrukcí s operandy závislými na tajných hodnotách. Tento model úniku je specifický pro některé architektury procesorů a instrukce, jejichž provedení trvá proměnnou dobu, viz například násobení s proměnnou dobou u některých procesorů s architekturou ARM. Tři uvedené modely úniku se obvykle používají společně. Jak model větvením, tak model úniku přístupem do paměti mají verzi, která je modifikuje tak, aby byla správně modelována existence vyrovnávací paměti (cache) procesoru. To například umožňuje kód, který přistupuje do paměti na základě tajné hodnoty, ale pouze v prostoru jednoho řádku vyrovnávací paměti (někdy v důsledku použití protiopatření pro před-načítání do této vyrovnávací paměti), takže útočník nezíská žádné tajné informace.

Pro statické nástroje jsou zásadní vlastnosti *korektnosti* (sound analysis) a *úplnosti* (completeness analysis) a jejich zjištění je dosažitelné. Korektní nástroj považuje za bezpečné pouze bezpečné programy bez úniku informací (vzhledem k danému modelu úniku), a nemá tedy falešně negativní výsledky (program s únikem dat), zatímco úplný nástroj považuje za nezabezpečené pouze nezabezpečené programy, a nemá tedy falešně pozitivní výsledky (programy bez úniku dat prohlášený nástrojem za nezabezpečený). V případě dynamických nástrojů je korektnost a úplnost často nedosažitelná a lze pouze uvažovat o míře falešně negativních a falešně pozitivních výsledků nebo obecně o třídách zranitelností, které nástroj může najít, a o třídách programových konstrukcí, které nástroj označí falešně.
S pojmem chyb ve výstupu nástroje souvisí jeho *flexibilita*, pokud jde o to, jaké hodnoty považuje za tajné. Některé nástroje dávají vývojáři možnost odtajnit tajnou hodnotu, a tím ji vyjmout z analýzy. To je zjevně dvousečná zbraň. Pro některé konstrukce programů je sice nezbytná, ale její zneužití vývojářem může vést k falešně negativním výsledkům.
Kryptografické implementace představují pro (statické) nástroje pro verifikaci programů komplikovaný problém na řešení. Při kryptografických velikostech vstupů a výstupů může být stavový prostor (s nímž nástroje pro verifikaci programů často pracují) pro nástroje příliš velký, a to i při použití mnoha jejich analytických triků. Důležitým kritériem je tedy *výkonnost* ovlivňující, zda je nástroj prakticky použitelný na reálných programech, např. kryptografických knihovnách.
V neposlední řadě jde i o celkovou *použitelnost* nástroje. Použitelnost je obtížné charakterizovat, protože obsahuje prvky všech dosud diskutovaných kritérií. Například pokud se jako vstupní úroveň nástroje použije doménově specifický jazyk (DSL), stávající projekty jej pravděpodobně nepřijmou, protože bude vyžadovat přepsání části jejich kódu. Vzhledem k tomu, že tyto nástroje jsou často produktem výzkumu, platí také obvyklé rčení o kvalitě výzkumného kódu a související připravenosti na profesionální použití. Navíc při absenci řádného balíčkování zůstávají nástroje a jejich závislosti často zastaralé a nefungují na aktuálních verzích svých knihovních závislostí. Praxe ukazuje, že použitelnost je významným problémem a překážkou pro širší přijetí analyzačních nástrojů.

2.1.2 Přehled nástrojů

Přehled v současnosti obsahující 40 různých nástrojů je zachycen na Obrázku 1 a je také dostupný na adrese `https://crocs-muni.github.io/ct-tools/`. Systematické vyhodnocení a praktické odzkoušení nástrojů je časově velmi náročná činnost a je plánováno na více navazujících etap projektu. Detailní vyhodnocení, podle kritérii z předešlé sekce, je v rámci Etapy 4 dostupné pro několik nejpopulárnějších nástrojů.
Nástrojů pro ověřování odolnosti kryptografických implementací proti časovým útokům existuje celá řada, ale zdá se, že téměř žádný z nich se nepoužívá automatizovaně mimo akademické publikace, které jej představily. To je znepokojující, protože jejich dopad je pak omezen na několik vybraných implementací, které se autoři rozhodli v dané publikaci jednorázově ověřit, zatímco reálné kryptografické knihovny se mění denně a nemají žádné automatizované ověřování.
Z analyzovaných statických nástrojů vynikají nástroje *ct-verif* a *SideTrail*, které jsou aktivně nasazeny v kryptografické knihovně Amazon s2n. Oba nástroje jsou však už zastaralé a neudržované. Za zmínku stojí také nástroj *Binsec/Rel* pro svůj přístup na binární úrovni a aktivní údržbu.
Použitelnost dynamických nástrojů je obvykle mnohem lepší než u statických nástrojů, přičemž přístup nástrojů *ctgrind*/*timecop* je z hlediska integrace téměř bez nákladů, protože postačují běžné testy programu nebo fuzzing kombinovaný se známými a udržovanými nástroji ze sady



Name	Year	Target	Technique	Guarantees
ABPV13	2013	C	Formal	sound
ANABLEPS	2019	Binary	Dynamic	no
Binsec/Rel	2020	Binary	Symbolic	sound with restrictions
Blazer	2017	Java	Formal	sound
BPT17	2017	C	Symbolic	sound with restrictions
CacheAudit	2013	Binary	Formal	other
CacheD	2016	Trace	Symbolic	no
CacheS	2019	Binary	Formal	sound with restrictions
CacheFix	2018			
COCO-CHANNEL	2018	Java	Symbolic	sound
Constantine	2021			
ctgrind	2010	Binary	Dynamic	sound with restrictions
ct-fuzz	2020	LLVM IR	Dynamic	no
ct-verif	2016	LLVM IR	Formal	sound
CT-WASM	2019	WASM	Formal	sound
DATA	2018	Binary	Dynamic	sound with restrictions
dudect	2017	Binary	Statistical	no
ENCIDER	2022	LLVM	Symbolic	sound with restrictions
ENCoVer	2012	Java	Formal	sound
FaCT	2019	DSL	Formal	sound
FlowTracker	2016	LLVM IR	Formal	sound
haybale-pitchfork	2019	LLVM IR	Symbolic	sound with restrictions
KMO12	2012	Binary	Formal	other
MemSan	-	LLVM IR	Dynamic	sound with restrictions
MicroWalk	2018	Binary	Dynamic	sound with restrictions
pitchfork-angr	2019			
SC-Eliminator	2018	LLVM IR	Formal	sound
SideTrail	2018	LLVM IR	Formal	other
Themis	2017	Java	Formal	sound
timecop	-	Binary	Dynamic	sound with restrictions
tis-ct	-	C	Symbolic	sound with restrictions
TriggerFlow	2018			
VirtualCert	2014	x86	Formal	sound

Obrázek 1: Přehled aktuálně pokrytých nástrojů a ukázka vyhodnocení pro jeden z nich.

Valgrind. Nástroj *dudect* je možné použít i v kontinuální integraci, pokud jsou pro něj vytvořeny speciální testovací funkce. Nástroje *MicroWalk*/*DATA* jsou dosti podobné a mohly by být vhodnější pro interaktivní testování implementací.

2.1.3 Další informace

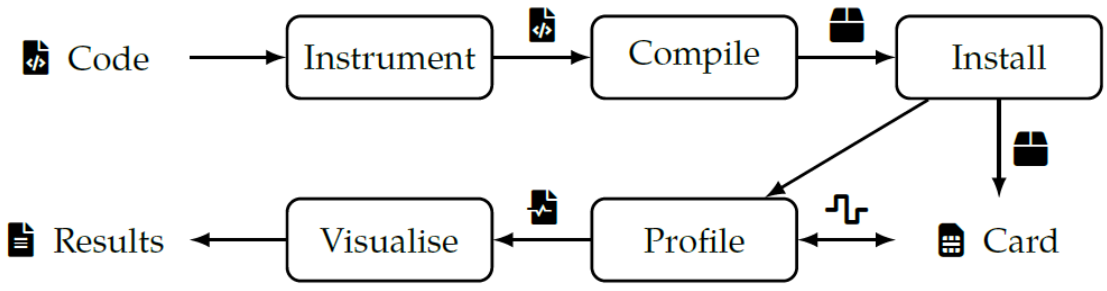
Přehled analyzovaných nástrojů je dostupný a postupně rozšiřovaný na adrese `https://crocs-muni.github.io/ct-tools/`.

2.2 *JCProfilerNext*: analýza časových závislostí JavaCard aplikací

Znalost doby vykonávání operací kódu pro různá vstupní data je důležitá informace pro zlepšení výkonu implementace (využito primárně vývojáři aplikací) i ověření odolnosti vůči úniku informací časovým postranním kanálem. Přesné měření rychlosti provádění dílčích operací na kryptografických čipových kartách zároveň není problém s jednoduchým řešením. Na rozdíl od běžného programu běžícího na PC není možné využít techniky typu monitorování stavu zásobníku volání nebo instrumentace kódu používané běžnými výkonnostními analyzátory, neboť čipová karta neumožňuje za běhu externím programem ani přistupovat k zásobníku, ani získat přesný čas přímo na kartě.
Výrobci čipových karet typicky nabízí svoje proprietární nástroje pro analýzu doby vykonávání operací pro jimi vyrobenými kartami. Tato řešení však mají řadu omezení – především je nelze použít karty jiných výrobců, nejsou volně dostupné a typicky nepodporují dostatečně detailní měření pro detekci časových odchylek. V roce 2017 tak byl vytvořen otevřený nástroj JCProfiler [7], který poskytoval možnost měření doby trvání jednotlivých řádků kódu pro JavaCard applety na libovolných kartách. Tento nástroj ale vyžadoval správné provedení řady manuálních kroků, neposkytoval vizualizace výsledků a neprováděl testování závislé na typu vstupních dat.
V rámci tohoto projektu byl vytvořen nástroj JCProfilerNext, který je kompletním přepisem původního nástroje JCProfiler, využívá plnou instrumentaci kódu pomocí analyzátoru Spoon[1] a odstraňuje jeho omezení. Navíc přidává funkčnost umožňující analyzovat spotřebu paměti a použití různých funkcí a algoritmů z JavaCard knihoven. Nástroj bude dále rozvíjen pro podporu detekce kódu s nekonstantní dobou vykonání – iniciální testování proběhlo s vybranými funkcemi knihovny JCMathLib (viz. sekce 2.3).

2.2.1 Princip fungování nástroje

Nástroj JCProfilerNext provádí analýzu cílového appletu v několika fázích. Zdrojový kód appletu je nejprve instrumentován dodatečnými měřícími funkcemi, zkompilován, zkonvertován a nahrán na kartu. Obslužná aplikace následně zasílá vstupní data pro profilování cílové metody, kontroluje místo jejího ukončení a měří dobu jejího trvání. Výsledná časová měření jsou zpracována a vizualizována. Celý proces je zobrazen na Obrázku 2, ukázka instrumentace pomocí automaticky vkládaných "pastí" s podmíněnými výjimkami je zobrazena na Obrázku 3.



Obrázek 2: Základní schéma funkčnosti nástroje JCProfilerNext.

[1] https://spoon.gforge.inria.fr/

```
// next fatal performance trap
public static short m_perfStop = 2;

public void test(short a) {
    if (m_perfStop == 1)
        // after the throw, measure and store elapsed time to
        // reach trap 1
        ISOException.throwIt((short) 1);

    short b = a + 10;
    if (m_perfStop == 2)
        // after the throw, measure elapsed time to reach trap 2
        // and store trap 2 - trap 1
        ISOException.throwIt((short) 2);

    b = a + 20;
    if (m_perfStop == 3)
        // after the throw, measure elapsed time to reach trap 3
        // and store trap 3 - trap 2
        ISOException.throwIt((short) 3);

    ...
}
```

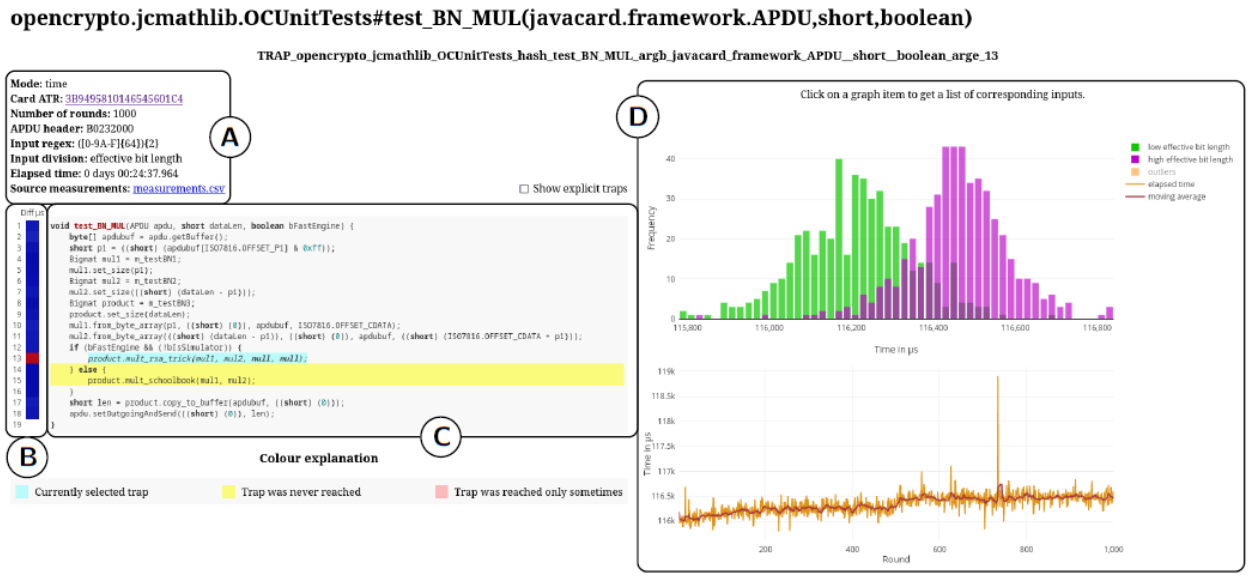
Obrázek 3: Ukázka principů fungování instrumentace kódu pomocí výjimkových pastí (traps) a jejich měření. Zobrazené řádky kódu jsou automaticky doplněny podmíněnými výjimkami (*if (m_perfStop...)*), díky kterým je možné postupným nastavováním hodnoty *m_perfStop* určit koncové místo, po které bude kód vykonán. Odečtením doby trvání sousedních pastí lze získat dobu trvání daného řádku kódu.

Nástroj lze využít v těchto základních módech:

1. **Měření času** – je provedeno měření času trvání všech jednotlivých řádků kódu zkoumané funkce pomocí časomíry na obslužném počítači.
2. **Měření paměti** – je provedeno měření spotřeby transientní (RAM) a perzistentní (EEPROM/flash) paměti pro všechny objekty alokované v konstruktoru appletu.
3. **Statistika využití funkcí a algoritmů** – je provedena detekce všech využívaných javacard balíků i konkrétních tříd a konstant.
4. **Personalizovatelný** – tento mód umožňuje vložit libovolný kus kódu definovaný vývojářem namísto standardních podmíněných výjimek používaných při běžné časové analýze. Lze jej použít například pro vložení operací výrazně viditelných prostřednictvím odběrové analýzy (tzv. oddělovače, např. volání *TRNG.getRandom()*), které následně umožní extrahovat délku oblasti mezi dvěma oddělovači a poskytnou výrazně přesnější měření doby jednotlivých operací v kódu.

2.2.2 Detekce kódu s variabilní dobou trvání

Možnost měření doby trvání kódu umožňuje testovat dobu trvání operace pro různá vstupní data a detekovat tak nežádoucí závislost dobu běhu operace na tajné informaci (typicky kryptografickém klíči). Detailní testování nástroje pro toto použití teprve připravujeme, Obrázek 4



Obrázek 4: Ukázka stránky zobrazující výsledek analýzy cílové funkce (C) pro konkrétní čipovou kartu (A) z hlediska časové náročnosti (B) jednotlivých řádků kódu, histogram doby trvání a dílčí měření pro různé vstupní data (D). Histogramy různé barvy zachycují měření s různou strukturou vstupních dat (např. různá efektivní bitová délka zpracovávaného vstupu).

ale již zachycuje ukázku výstupu pro analýzu metody pro násobení velkých čísel BN_MUL() z knihovny JCMathLib při zpracování dvou skupiny vstupů s malou resp. velkou efektivní bitovou délkou operandu. Histogram pro obě skupiny jasně ukazuje přítomnost časové závislosti.

2.2.3 Další informace

Detailní popis vyvinutého nástroje a vyhodnocení je dostupné (v anglickém jazyce) v diplomové práci Lukáše Zaorala "Automatic Performance Profiler for Security Analysis of Cryptographic Smart Cards" [9]. Zdrojové kódy jsou dostupné v repositáři `https://github.com/lzaoral/JCProfilerNext`.

2.3 *JCMathLib*: matematická knihovna pro platformu JavaCard

JCMathLib je knihovna s otevřeným zdrojovým kódem pro platformu JavaCard, která uživateli poskytuje základní matematické operace potřebné pro implementaci kryptografických algoritmů. Na rozdíl od proprietárních rozhraní poskytovaných výrobci konkrétních zařízení, knihovna JCMathLib závisí pouze na otevřeném rozhraní platformy JavaCard, a tudíž je použitelná na široké řadě karet využívajících tuto platformu. Za tyto vlastnosti se platí nižší rychlostí a odolností vůči útokům postranními kanály. Knihovnu tedy nedoporučujeme používat pro produkční nasazení v bezpečnostně kritických aplikacích, ale naopak je vhodná pro vývoj.
Knihovna byla původně vyvíjena převážně v roce 2018 a nabízela podporu pro modulární aritmetiku a operace na eliptické křivce. Od té doby byly vydány nové karty s platformou JavaCard verze 3.0.5, podporující algoritmy `ALG_EC_SVDP_DH_PLAIN_XY` a `ALG_EC_PACE_GM`, které umožňují efektivnější implementaci operací s body na eliptické křivce. Součástí změn provedených v této části práce bylo zakomponování těchto operací do knihovny JCMathLib. Pro volbu, které operace se mají použít na konkrétním zařízení byly přidány profily karet.

2.3.1 Podpora pro různé typy použitých karet

Profily karet umožňují uživateli specifikovat, které algoritmy daná karta podporuje a knihovna JCMathLib na základě této konfigurace rozhoduje, jakým způsobem se budou jednotlivé operace vyhodnocovat. Možnost konfigurovat profily je důležitá nejen pro provozování nejnovějších algoritmů, ale i proto, že implementace operací poskytovaných JavaCard rozhraním se mohou na různých kategoriích vstupů lišit mezi různými kartami a ne vždy podporují všechny přípustné vstupy. Neboť některé implementace v knihovně JCMathLib mohou na tomto specifickém chování záviset, bez jejich podpory nemusí výpočty probíhat korektně, a tedy je potřeba toto chování také specifikovat v profilech karet. Aktuálně máme definované profily pro 3 typy fyzických čipových karet a simulátor jcardsim.org (viz Tabulka 1), ale tento seznam se dalším testováním rozrůstá.

Tabulka 1: Aktuální předkonfigurované profily čipových karet s povolenými operacemi. Nové profily karet budou přibývat s dalším testováním.

	Simulator	J2E145G	JCOP3	JCOP4
RSA_MULT_TRICK	✗	✓	✓	✓
RSA_MOD_EXP	✓	✓	✗	✗
RSA_PREPEND_ZEROS	✓	✗	✗	✗
RSA_KEY_REFRESH	✓	✗	✗	✗
EC_HW_XY	✓	✗	✓	✓
EC_HW_X	✓	✓	✓	✓
EC_HW_ADD	✓	✗	✗	✓
EC_SW_DOUBLE	✓	✗	✗	✗
DEFERRED_INITIALIZATION	✗	✗	✗	✓

2.3.2 Podpora podpisů dle Ed25519

Od verze 3.1.0 čipové karty s platformou JavaCard podporují vytváření moderních podpisů typu Ed25519, ale karty podporující tento typ podpisu stále nejsou rozšířené. Díky JCMathLib je

možné implementovat tento typ podpisů i na starších čipových kartách, což jsme demonstrovali implementací appletu JCEd25519[2]. Tato implementace vyžadovala další rozšíření knihovny JCMathLib, která do ní byla také zakomponována. Implementace JCEd25519 také obsahuje návrh delegování jedné výpočetně náročné operace na řídící zařízení bez potřeby dodatečné důvěry, což umožňuje podstatně zrychlit výpočet na starších čipových kartách.

2.3.3 Podpora komprimovaných bodů ECPoint, další vylepšení

Některé kryptografické algoritmy a protokoly potřebují pracovat s body, které jsou dodány na zařízení v komprimované formě, ale implementace JavaCard platformy mnohdy neumožňuje body z této formy rekonstruovat. Tato rekonstrukce byla v rámci rozšíření funkcionality také doplněna, avšak je výpočetně náročná, takže pokud je to možné, měla by být provedena na řídícím zařízení. Současně s touto změnou byly implementovány pomocné funkce, poskytující informace o konkrétním bodu na eliptické křivce, například, sudost Y-souřadnice.
Pro zefektivnění dalšího vývoje byla knihovna JCMathLib portována na build systém Gradle a část kódu refaktorována. Tyto změny umožňují snazší testování, měření pokrytí kódu testy. Jednotkové testy byly rozšířeny pro testování nově přidaných operací.

2.3.4 Další informace

Původní verze knihovny je dostupná v repositáři `https://github.com/OpenCryptoProject/JCMathLib`. Výše popsané úpravy jsou v současné době umístěny v repositáři `https://github.com/dufkan/JCMathLib` a po dokončení relevantních úprav budou sloučeny s hlavním repositářem.

[2] `https://github.com/dufkan/JCEd25519`

2.4 *ec-detector*: detekce eliptických křivek ve zdrojovém kódu

Bezpečnost kryptografie eliptických křivek (ECC) je založena na vhodné volbě eliptické křivky. V případě klasických protokolů ECC, jako jsou například ECDSA, ECDH nebo EdDSA, se pro reálné aplikace používá několik málo křivek. Nástupem protokolů využívajících párování na křivkách se situace změnila. Desítky tzv. pairing-friendly křivek byly navrženy pro reálné využití [2, 4], jejich počet každým rokem stoupá a současný stav, včetně jejich bezpečnosti, je nepřehledný. Softwarové dokumentace často nenabízejí uspokojivou odpověď na otázku podporovaných křivek. EC-detector je jednoduchý nástroj na procházení zdrojových kódů a prohledávání podporovaných křivek na základě klíčových slov a konstant.
ec-detector je rozšíření již existujícího nástroje *crypto-detector* [6] přidávající vyhledávání křivek. Klíčové slova pro vyhledávání (jména, identifikátory a definující konstanty křivek) se čerpají z naší databáze všech používaných standardních křivek nástroje DiSSECT [5]. Výstupem je seznam nalezených křivek s úryvkem obsahující nalezená klíčová slova, které lze poté projít a zkontrolovat. Vstupem je libovolná cesta ke zdrojovým kódům včetně například odkazu do vzdáleného repositáře.
Databáze DiSSECT obsahuje v současnosti přes 200 eliptických křivek včetně jejich základních vlastností a konstant. DiSSECT se nezaměřuje pouze na populární křivky, ale snaží se zahrnout všechny, které byly nebo jsou doporučeny pro kryptografické použití. Zejména pokrývá všechny Edwardsovy (25), Montgomeryho (5), Weierstrassovy (121) křivky a binární (44), prvočíselné (151) a extension (4) křivky. Dále databáze čítá 31 pairing-friendly křivek včetně všech základních rodin (MNT, BLS, BN). Zdrojem velkého počtu všech křivek jsou oficiální standardy ANSI X9.62, NIST: FIPS 186, SECG, Brainpool, ANSSI, GOST R, IEEE, ISO, OSCAA.

2.4.1 Ukázka použití

Nástroj ec-detector lze snadno použít vůči adresáři nebo repositáři se zdrojovými kódy:

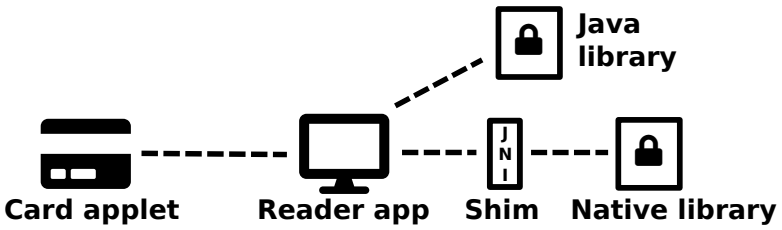
```
$ python ecdetector.py https://github.com/zcash/zcash
...
Curves found (more info in /ec-detector/zcash-zcash-master.ec):
['secp256k1', 'Ed25519', 'Curve25519', 'Pallas', 'JubJub', 'Ed448', 'BLS12-381',
'Vesta']
```

2.4.2 Další informace

Nástroj je dostupný v repositáři `https://github.com/vojtechsu/ec-detector/`. Seznam všech hledaných křivek je poskytnut projektem DiSSECT dostupným na adrese `https://dissect.crocs.fi.muni.cz/`.

2.5 *ECTester*: testování implementací kryptografie eliptických křivek

ECTester[3] je nástroj na black-box testování implementací kryptografie eliptických křivek. Black-box testování nevyžaduje přístup ke kódu implementace, či jiné znalosti o detailech implementace, pouze přístup k veřejnému rozhraní kryptografické implementace. Nástroj je schopen testovat implementace v kryptografických knihovnách i čipových kartách založených na populární platformě JavaCard. Jeho architektura je představena na Obrázku 5. Testy jsou založeny na známých útocích, jako jsou *invalid-curve* útoky, *small-subgroup* útoky, nebo *degenerate-curve* útoky a dále se zaměřují na možné chyby v širším pohledě a to ve formě testů s chybným formátem podpisů, testů pro hraniční hodnoty inspirované zranitelnostmi z minulosti, a podobně. Detaily rozsáhlé testovací sady, obsahující 11 podkategorií testů, jsou dostupné na `https://github.com/crocs-muni/ECTester/blob/master/docs/TESTS.md`.



Obrázek 5: Architektura nástroje ECTester.

Nástroj lze spouštět pro tři základní typy zařízení: 1) simulátor čipových karet jcardsim.org, 2) fyzická čipová karta s platformou JavaCard a 3) vybraná kryptografická softwarová knihovna. Sady testů byly použity na otestování 12 karet JavaCard s podporou kryptografie eliptických křivek a 12 populárních kryptografických knihoven s otevřeným zdrojovým kódem:

- BoringSSL 1.1.1
- Botan 2.12.1
- BouncyCastle Security Provider v1.58
- Crypto++ 8.3.0
- ipp-crypto 2021.3.0
- libgcrypt 1.8.5
- LibreSSL 3.4.1
- libtomcrypt 1.18.2
- Mbed TLS 2.16.3
- Nettle 3.5
- OpenSSL 1.1.1f
- Sun Elliptic Curve provider 1.8

Toto testování odhalilo několik zranitelností ke kterým bylo přiřazeno číslo CVE, jejich seznam je dostupný na `https://github.com/crocs-muni/ECTester/blob/master/docs/VULNS.md`. Mimo výsledků ve formě zveřejněných zranitelností výsledky obsahují také několik zajímavých nálezů z analýzy kryptografických knihoven [3], kdy některé testy dovedly knihovny do nekonečného cyklu nebo vedly k ukončení procesu segmentační chybou (SEGFAULT) v kódu knihovny. Testy rovněž odhalily, že některé knihovny mohou být zranitelné vůči invalid-curve či small-subgroup útokům, pokud jejich používatelé nepoužijí dodatečné validační funkce, které knihovny sice poskytují, ale nevyžadují a neprovádí automaticky.

[3] `https://github.com/crocs-muni/ECTester`

2.5.1 Ukázka použití: simulátor jcardsim.org

Spustí základní testy na simulátoru čipových karet s JavaCard platformou.

```
$ ECTesterReader.jar -t -s
```

2.5.2 Ukázka použití: fyzická karta s JavaCard

Instalace testovacího appletu na kartu:

```
$ gp --install ectester222.cap -d
```

Kontrola korektního nainstalování:

```
$ ECTesterReader.jar -nf
```

```
Card ATR:     3bfc18000081318045906746a00641606000000001e
Card protocol:                          T=1
ECTester applet version:                v0.3.3
ECTester applet APDU support:           extended length
JavaCard API version:                   3.0
JavaCard supports system cleanup:       true
Array sizes (apduBuf,ram,ram2,apduArr): 261 256 256 1024
   ...
```

Spuštění na cílové kartě, sběr výsledků do souboru out.csv:

```
$ ECTesterReader.jar --ecdsa 3 -fp -b 256 -u --fixed --time-unit nano -o out.csv
```

Formát výstupu out.csv:

```
index;signTime[nano];verifyTime[nano];data;pubW;privS;signature[SHA1];nonce;valid
```

Spuštění vybrané sady testů (zde např. sada *invalid*):

```
$ ECTesterReader.jar --fixed --time-unit nano -o out_invalid.csv --test invalid
```

2.5.3 Ukázka použití: vybraná softwarová kryptografická knihovna

Výpis dostupných kryptografických knihoven v systému (dostupné jako Java CSP):

```
$ ECTesterStandalone.jar list-libs
```

```
- Sun Elliptic Curve provider (EC, ECDSA, ECDH)
- Version: 1.800000
- Supports native timing: []
- KeyPairGenerators: EC
- KeyAgreements: ECDH
- Signatures: NONEwithECDSA, ECDSA, SHA384withECDSA, SHA224withECDSA...
- Curves: X9.62 c2tnb191v1, X9.62 c2tnb191v2, X9.62 c2tnb191v3, X9.62 c2tn...

- BouncyCastle Security Provider v1.58
- Version: 1.580000
- Supports native timing: []
- KeyPairGenerators: ECMQV, ECDSA, EC, ECDH, ECDHC
- KeyAgreements: ECCDHwithSHA384KDF, ECDHwithSHA256KDF, ECDHwithSHA384KDF...
- Signatures: SHA1withCVC-ECDSA, NONEwithECDSA, ECGOST3410, SHA256withECNR...
```

```
- Curves: B-163, B-233, B-283, B-409, B-571, FRP256v1, K-163, K-233, K-283...
    ...
```

Zjištění řetězce názvu cílové knihovny (např. “BouncyCastle Security Provider v1.58”)

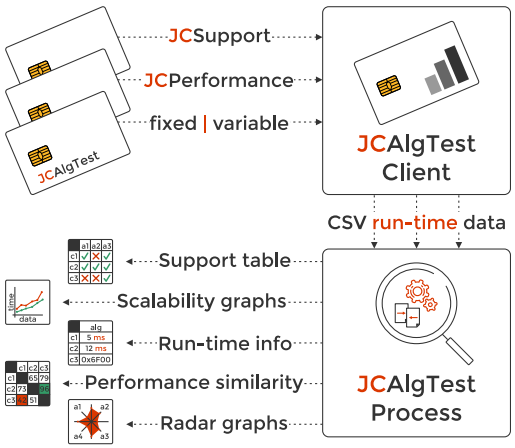
Provedení operace a sběr výstupů (zde 3 opakování):
`$ ECTesterStandalone.jar ecdh -n 3 -cn secp256r1 "BouncyCastle Security Provider v1.58"`
`$ ECTesterStandalone.jar ecdsa -n 3 -t NONEwithECDSA -nc secg/secp256r1 --fixed "BouncyCastle Security Provider v1.58"`
Spuštění vybrané sady testů (zde sada *cofactor*):
`$ ECTesterStandalone.jar test cofactor "BouncyCastle Security Provider v1.58"`

2.5.4 Další informace

Zdrojové kódy nástroje jsou dostupné v repositáři `https://github.com/crocs-muni/ECTester`. Vizualizované výsledky pro dosud analyzované karty a softwarové knihovny jsou dostupné na `https://crocs-muni.github.io/ECTester/`.

2.6 *JCAlgTest*: Sběr a analýza metadat čipových karet

V rámci projektu pokračujeme v rozvoji otevřeného nástroje JCAlgTest [1], který již od roku 2007 umožňuje sběr vlastností implementací kryptografických čipových karet s platformou JavaCard. Měřené vlastnosti produkované nástrojem JCAlgTest jsou sbírány do otevřené databáze a v současnosti pokrývají více než 100 různých čipových karet (zhruba polovina pochází z karet dostupných v laboratoři CRoCS, zbytek je poskytnut uživateli nástroje). Základní schéma provádění sběru dat je zobrazeno na Obrázku 6.



Obrázek 6: Základní schéma provádění sběru dat o vlastnostech cílové čipové karty a zpracování výsledků nástrojem JCAlgTest.

Získané výsledky mají tyto základní možnosti využití:

1. **Výběr karty podporující požadovaný algoritmus** (např. SHA-2), typicky využito vývojářem hledajícím vhodnou čipovou kartu.
2. **Výběr karty poskytující dostatečný výkon či paměť** (např. rychlost vytváření digitálních podpisů ECDSA), typicky využito vývojářem hledajícím vhodnou čipovou kartu.
3. **Porovnání podobnosti karet**, typicky využito bezpečnostním administrátorem odpovědným za počáteční analýzu a následnou kontrolu shodnosti očekávaných a obdržených karet.
4. **Analýza vlastností kryptografických klíčů generovaných kartou**, typicky využito bezpečnostních výzkumníkem ověřujícím (ne-)přítomnost zranitelnosti.
5. **Analýza trendů vývoje ekosystému čipových karet**, typicky využito pro získání hlubšího vhledu do dynamiky směřování celého ekosystému z hlediska podporovaných kryptografických algoritmů, funkcí a jejich rozšíření mezi kartami od různých výrobců.

2.6.1 Implementované moduly

Nástroj JCAlgTest je implementován primárně v jazyce Java (obslužný klient spouštěný na PC) resp. JavaCard (applet nahraný na analyzovanou čipovou kartu), doplněný o různorodé analytické nástroje vyvíjené dle potřeby a obsahuje moduly pro sběr a vyhodnocení následujících vlastností:

1. **Identifikátory ATR a CPLC, dostupná paměť** – modul poskytuje rychlou základní identifikaci karty a umožňuje seskupit karty dle jejich společných prvků jako čip nebo operační systém (ICFabricator, ICType, OSID, ICFabDate, viz. Obrázek 7). Kompletní graf karet se dostupným CPLC (Card Product Lifecycle data) je zachycen na Obrázku 8).

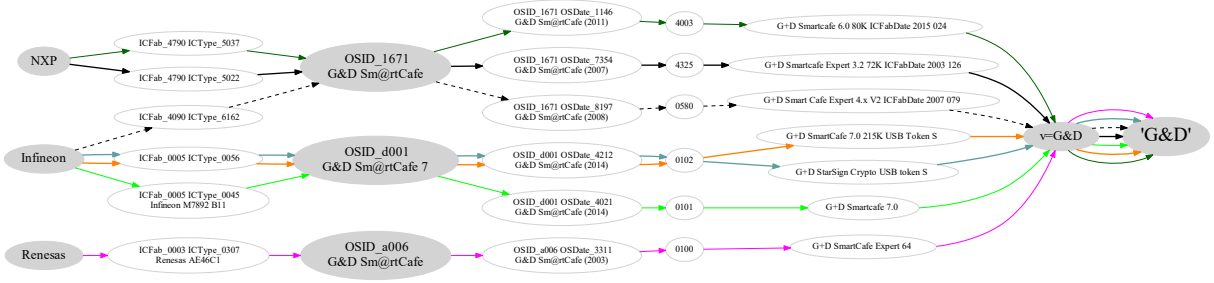
2. **Podporované JavaCard algoritmy** – modul ověřuje dostupnost implementace pro všechny konstanty algoritmů extrahované ze specifikace JavaCard API v2.0 (vydaná v roce 2000) až do po specifikaci v3.1 (vydaná v roce 2019) a celkově pokrývá více než 360 kryptografických algoritmů, délek klíčů a možností doplňování (padding). Vyhodnocení míry podpory algoritmů přes všechny karty v databázi je zachyceno v Tabulce 2.

3. **Rychlost implementovaných algoritmů pro různé délky dat** – modul poskytuje detailní informaci o rychlosti zpracování dat všemi dostupnými algoritmy pro různé délky vstupních dat. Umožňuje zjistit očekávanou dobu běhu pro komplexnější kombinované operace, porovnat karty různých výrobců i detekovat interní velikosti vyrovnávacích paměti způsobující výraznější nárůst času zpracování pro větší délky vstupních dat. Celkově je měřeno více než 2300 kombinací {algoritmus, metoda, délka_dat} s rozlišením lepším než 1 milisekunda i přes nepřítomnost časového měření přímo na kartě.

4. **Instalované javacard balíky (packages)** – modul provádí detekci pro 89 balíků java.*, javacard*.*, org.globalplatform.* a visa.openplatform.* a jejich verzí pomocí speciálně modifikovaného instalačního JavaCard *.cap souboru. V případě neexistence konkrétního balíku na kartě instalace selže.

5. **Generování RSA a ECC klíčů a podpisů** – modul umožňuje generovat velké množství RSA a ECC klíčů a digitálních podpisů přímo na kartě a provede jejich export na PC pro další zpracování.

6. **Prezentace výsledků** – modul je odpovědný za vygenerování zpracovaných a vizualizovaných dat pro koncového uživatele.



Obrázek 7: Ukázka vzájemné závislosti různých položek IC Fabricator, IC Type, verze OS a data zveřejnění extrahovaných pro sedm různých karet vyráběných firmou Giesecke&Devrient. Jedna karta z databáze odpovídá uzlům spojených čárou stejné barvy a typu.

2.6.2 Využití pro certifikační účely

Certifikace kryptografických čipových karet dle Common Criteria nebo NIST FIPS 140-2 je široce používaný proces, během kterého hodnotící laboratoř ověřuje tvrzení výrobce o produktu

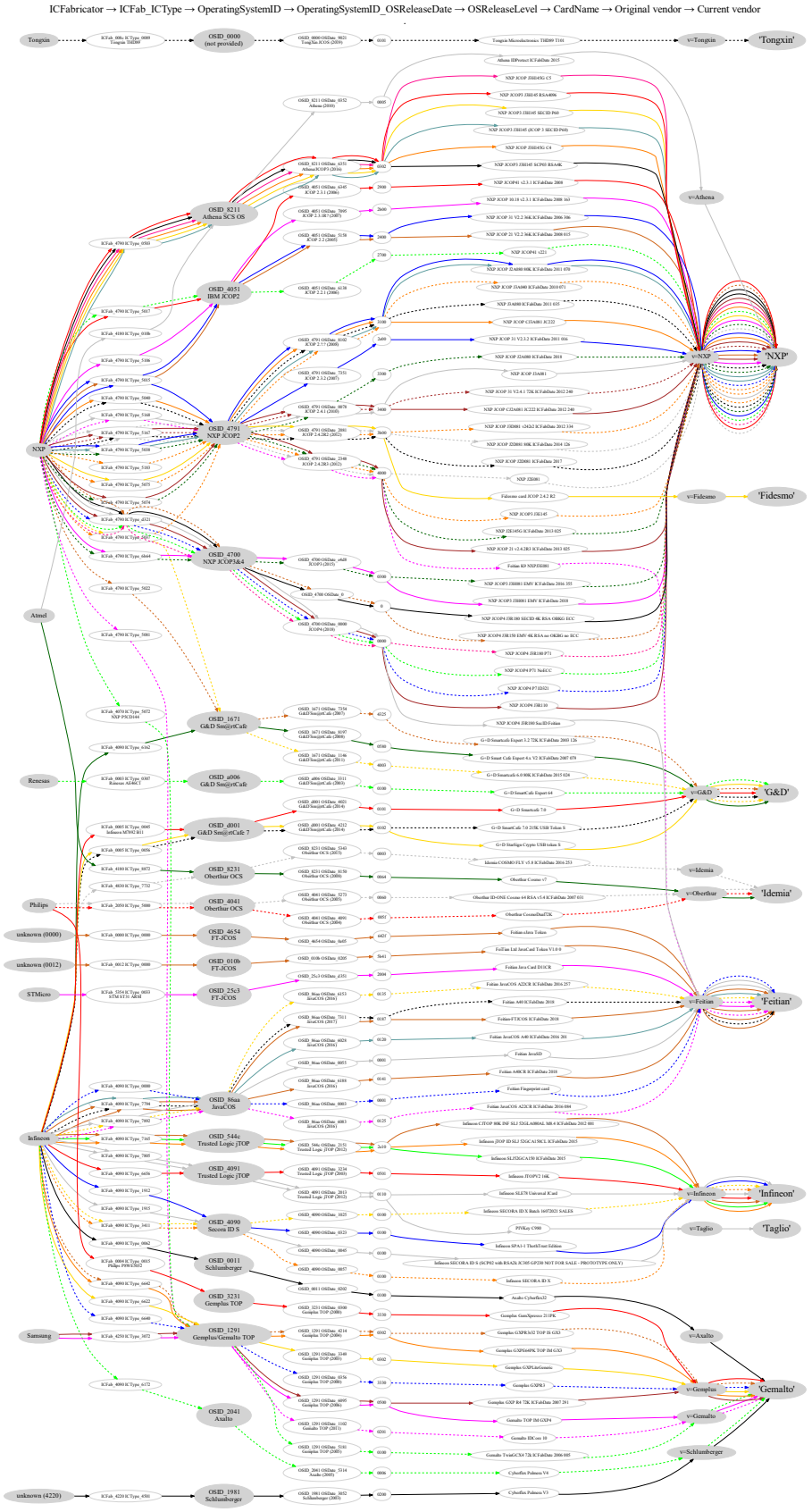
Feature	First in version	JC ⩽ 2.2.1 (21 cards)	JC 2.2.2 (26 cards)	JC 3.0.1/2 (12 cards)	JC 3.0.4 (29 cards)	JC 3.0.5 (11 cards)
Truly random number generator						
TRNG (ALG_SECURE_RANDOM)	⩽ 2.1	100%	100%	100%	100%	100%
Block ciphers used for encryption or MAC						
DES (ALG_DES_CBC_NOPAD)	⩽ 2.1	100%	100%	100%	100%	100%
AES (ALG_AES_BLOCK_128_CBC_NOPAD)	2.2.0	52%	96%	100%	100%	100%
KOREAN SEED (ALG_KOREAN_SEED_CBC_NOPAD)	2.2.2	5%	62%	75%	34%	0%
Public-key algorithms based on modular arithmetic						
1024-bit RSA (ALG_RSA(_CRT) LENGTH_RSA_1024)	⩽ 2.1	76%	96%	100%	93%	82%
2048-bit RSA (ALG_RSA(_CRT) LENGTH_RSA_2048)	⩽ 2.1	67%	96%	100%	93%	82%
4096-bit RSA (ALG_RSA(_CRT) LENGTH_RSA_4096)	3.0.1	0%	0%	0%	3%	0%
1024-bit DSA (ALG_DSA LENGTH_DSA_1024)	⩽ 2.1	5%	8%	8%	10%	0%
Public-key algorithms based on elliptic curves						
192-bit ECC (ALG_EC_FP LENGTH_EC_FP_192)	2.2.1	5%	62%	83%	66%	82%
256-bit ECC (ALG_EC_FP LENGTH_EC_FP_256)	3.0.1	0%	50%	75%	66%	82%
384-bit ECC (ALG_EC_FP LENGTH_EC_FP_384)	3.0.1	0%	12%	17%	62%	82%
521-bit ECC (ALG_EC_FP LENGTH_EC_FP_521)	3.0.4	0%	4%	8%	45%	82%
ECDSA SHA-1 (ALG_ECDSA_SHA)	2.2.0	24%	84%	100%	69%	82%
ECDSA SHA-2 (ALG_ECDSA_SHA_256)	3.0.1	5%	12%	100%	69%	82%
ECDH IEEE P1363 (ALG_EC_SVDP_DH)	2.2.1	29%	81%	100%	69%	82%
IEEE P1363 plain coord. X (ALG_EC_SVDP_DH_PLAIN)	3.0.1	5%	4%	67%	48%	82%
IEEE P1363 plain c. X,Y (ALG_EC_SVDP_DH_PLAIN_XY)	3.0.5	0%	0%	0%	17%	82%
Modes of operation and padding modes						
ECB, CBC modes	⩽ 2.1	100%	100%	100%	100%	100%
CCM, GCM modes (CIPHER_AES_CCM, CIPHER_AES_GCM)	3.0.5	0%	0%	0%	0%	0%
PKCS1, NOPAD padding	⩽ 2.1	95%	100%	100%	100%	100%
PKCS1 OAEP scheme (ALG_RSA_PKCS1_OAEP)	⩽ 2.1	14%	31%	8%	41%	82%
PKCS1 PSS sheme (ALG_RSA_SHA_PKCS1_PSS)	3.0.1	14%	19%	83%	41%	100%
ISO14888 padding (ALG_RSA_ISO14888)	⩽ 2.1	14%	12%	8%	0%	0%
ISO9796 padding (ALG_RSA_SHA_ISO9796)	⩽ 2.1	81%	100%	100%	86%	100%
ISO9797 padding (ALG_DES_MAC8_ISO9797_M1/M2)	⩽ 2.1	90%	100%	100%	100%	100%
Hash functions						
MD5 (ALG_MD5)	⩽ 2.1	90%	77%	92%	62%	0%
SHA-1 (ALG_SHA)	⩽ 2.1	95%	100%	100%	100%	100%
SHA-256 (ALG_SHA_256)	2.2.2	14%	88%	100%	97%	100%
SHA-512 (ALG_SHA_512)	2.2.2	5%	23%	25%	90%	100%
SHA-3 (ALG_SHA3_256)	3.0.5	0%	0%	0%	0%	0%

Tabulka 2: Míra podpory algoritmů specifikovaných v JavaCard API (JC API). Pro daný algoritmus je uvedena verze JC API, která jej první definovala. Následující sloupečky uvádí míru podpory algoritmu mezi kartami s nejvyšší podporovanou verzí získanou voláním $JCSystem.getVersion()$.

a příslušná autorita vydává produktový certifikát. V případě čipových karet je identifikace produktu prováděna typicky dle názvu, typu, řetězce zasílaného kartou po aktivaci (tzv. Answer To Reset) či údaji o životním cyklu výroby karty (tzv. CPLC). Přestože CPLC stačí ke spárování zakoupené karty s příslušným certifikátem při nákupu od důvěryhodného prodejce, lze s takovými statickými metadaty uloženými na kartě snadno manipulovat. Různí prodejci třetích stran nabízejí možnost libovolně personalizovat ATR i CPLC, což ztěžuje posouzení typu a zabezpečení karty koncovým uživatelem.
Nástroj JCAlgTest umožňuje rozšířit v současnosti používanou identifikaci karet o popisnější sadu metadat extrahovaných přímo z podporovaných funkcí, profilování výkonu a vlastností generovaných kryptografických klíčů. Tato metadata mohou být získány přímo hodnotící laboratoří, připojena k certifikátu a později ověřena koncovým uživatelem, často bez nutnosti speciálních znalostí a vybavení, což vede ke zvýšení důvěry v původ zakoupeného produktu.

2.6.3 Další informace

Detailní popis nástroje a vyhodnocení je dostupné (v anglickém jazyce) v článku “JCAlgTest: Robust identification metadata for certified smartcards” [8] vytvořeném a prezentovaném v roce 2022 v rámci řešení tohoto projektu. Naměřené datové soubory i zpracované výsledky jsou dostupné na webu `http://jcalgtest.org`, zdrojové kódy jsou dostupné v repositáři `https://github.com/crocs-muni/JCAlgTest/`.



Obrázek 8: Skupiny karet seskupené dle relevantních položek extrahovaných z CPLC informací (celkově 71 karet). Položky odpovídající konkrétní kartě jsou spojeny čárou stejné barvy a typu.

3 Závěr

Veškeré plánované činnosti v rámci Etapy 4 byly úspěšně dokončeny a proběhl sběr požadavků i prezentace průběžných výsledků aplikačnímu garantovi v rámci tří osobních setkání a workshopů (červenec, září a listopad 2022). V rámci Etapy 4 byly prováděny práce na pěti různých nástrojích a jedné přehledové databázi provádějících analýzu kryptografických primitiv vzhledem k postranním kanálům, vyhodnocení relevantních výstupů a implementaci vybraných kryptoprimitiv.

Přehled existujících nástrojů jiných autorů pro detekci časové nekonstantnosti implementací je poskytnut v průběžně rozšiřované databázi *ct-tools* (v současnosti obsahuje přes 40 nástrojů). Nástroj *ec-detector* provádí detekci použitých typů eliptických křivek v existujícím zdrojovém kódu využitelnou pro iniciální audit kryptografického kódu. Nástroj *JCProfilerNext* provádí analýzu rychlosti běhu kódu přímo na cílové kartě, paměťové náročnosti a používaných balíků JavaCard aplikací a umožňuje detekovat části časově náročné kódu (rychlostní optimalizace) a detekci časově nekonstantních částí kódu. Nástroj *ECTester* provádí test korektnosti implementace kódu využívajících kryptografii na bázi eliptických křivek a umožňuje testovat implementace na čipových kartách i softwarové kryptografické knihovny. Knihovna *JCMathLib* pro platformu JavaCard poskytuje implementace nízkoúrovňových objektů typu velká čísla (Bignat) nebo bod na křivce (ECPoint) a jejich související operace, které jinak nejsou dostupné ve standardním JavaCard API. Nástroj *JCAlgTest* poskytuje aplikaci pro sběr a analýzu funkčních i výkonnostních metadat získaných z čipových karet a související otevřenou databázi.

Na základě zpětné vazby od aplikačního garanta a náplně dalších etap bude v následujících fázích projektu probíhat další rozvoj relevantních nástrojů a jejich vyhodnocení.

Reference

[1] DEVELOPERS, JCAlgTest. *JavaCard Algorithm Test, Detailed analysis of cryptographic smart cards running with JavaCard platform.* http://jcalgtest.org/, 2022.

[2] GUILLEVIC, Aurore. *Pairing friendly curves.* https://members.loria.fr/AGuillevic/pairing-friendly-curves, 2021.

[3] HOFMAN, David. *Analysis of implementations of ECC libraries.* diploma thesis, Masaryk University, 2021. https://is.muni.cz/auth/th/bsc8v.

[4] SAKEMI, Yumi, KOBAYASHI, Tetsutaro, SAITO, Tsunekazu, and WAHBY, Riad. *Pairing friendly curves.* https://datatracker.ietf.org/doc/draft-irtf-cfrg-pairing-friendly-curves/, 2022.

[5] SEDLÁČEK, Vladimír, SUCHÁNEK, Vojtěch, and DUFKA, Antonín. *Distinguisher of Standard & Simulated Elliptic Curves via Traits.* https://dissect.crocs.fi.muni.cz, 2022.

[6] SLATER, Joe and KAMYARKAVIANI. *Crypto Detector.* https://github.com/Wind-River/crypto-detector, 2022.

[7] SVENDA, Petr. *JCProfiler – performance profiler for JavaCard code.* https://github.com/OpenCryptoProject/JCProfiler, 2018.

[8] SVENDA, Petr, KVASNOVSKY, Rudolf, NAGY, Imrich, and DUFKA, Antonin. *JCAlgTest: Robust identification metadata for certified smartcards.* In 19th International Conference on Security and Cryptography, pp. 597–604. Lisabon: INSTICC, 2022. ISBN 978-989-758-590-6. doi:10.5220/0000163500003283.

[9] ZAORAL, Lukas. *Automatic Performance Profiler for Security Analysis of Cryptographic Smart Cards.* diploma thesis (submitted), Masaryk University, 2022.

Příloha: Analýza rizik

Tabulka 3: Analýza rizik relevantních k Etapě 4.

Riziko	Možný dopad rizika	Skutečný dopad rizika	Datum rizika	Opatření pro minimalizaci/eliminaci
Omezení dostupnosti potřebných nízkoúrovňových primitiv na čipových kartách potřebných pro další rozvoj knihovny JCMathLib	Omezení nebo nemožnost implementovat kryptoprimitivum vyšší úrovně	Snížený výkon implementace kryptoprimitiva	-	Průběžné testování na kartách různého typu, zavedení profilů karet umožňující využívat alternativní (pomalejší) implementace, návrh nových možností implementace
Nedostatečné rozlišení časomíry na straně PC pro nástroj JCProfiler-Next	Nemožnost přesného měření doby trvání operací	Snížená přesnost měření	-	Vývoj podpory přesného měření času s využitím externího osciloskopu