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

Identifikační kód: VJ02010010

Odborná zpráva - Etapa 09

Období: 01/2023 - 12/2023

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:	Dr-Ing. Martin Novotný, Ph.D. Tel.: +420224358715 Email: martin.novotny@fit.cvut.cz

Obsah

1 Úvod **2**

2 Výzkum metod protiopatření eliminující útoky postranními kanály, SW prototypová implementace vybraných primitiv na jednočipy a a čipové karty **3**

2.1 Přehled frekvence podpory JavaCard algoritmů v softwarových implementacích s otevřeným zdrojovým kódem 4

2.1.1 JavaCard platforma v certifikovaných zařízeních 4

2.1.2 Analýza aktivity ekosystému s otevřeným zdrojovým kódem 5

2.1.3 Další informace 7

2.2 *JCMathLib 2.0*: matematická knihovna pro platformu JavaCard 9

2.2.1 Efektivnější akcelerace modulárního násobení pomocí RSA koprocesoru . 9

2.2.2 Změna zarovnání interní reprezentace typu BigNat 9

2.2.3 Alternativní implementace typu BigNat pro karty podporující typ `int` . . 10

2.2.4 Další informace 10

2.3 Bezpečná implementace ECC algoritmů na jednočipech MCU 11

3 Závěr **14**

Číslo a název aktivity: Etapa 09	
Období řešení:	01/2023 - 12/2023
Cíl aktivity:	Výzkum metod protiopatření eliminující útoky postranními kanály. SW implementace vybraných primitiv na čip. Návrh a implementace prototypu.
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 příloha článek popisující analýzu Javacard ekosystému (CARDIS 2023, EN), 3x GitHub repositář nástroje, knihovny a přehledové databáze. Rozšířená implementace knihovny JCMathLib.
Výsledky:	D - stať ve sborniku: Lukas Zaoral, Antonin Dufka, Petr Svenda. The adoption rate of JavaCard features by certified products and open-source projects. Proceedings of the 22nd Smart Card Research and Advanced Application Conference (CARDIS'23). (v procesu publikace pokonferenčního sborníku, DOI zatím nepřiřazeno).
Doložení splnění aktivity:	Odborná zpráva – Etapa 9
Shrnutí:	Všechny podmínky pro přechod do další aktivity byly splněny.

1 Úvod

V rámci etapy #9 (01/2023 - 12/2023) proběhla analýza a následný návrh a prototypová implementace kryptografických implementací pro oblast jednočipů a kryptografických karet.
Na základě diskuze s aplikačním garantem proběhla detailní analýza ekosystému JavaCard se zaměřením na dostupnost a použitelnost kryptografických funkcí na současných čipových kartách a jejich vývoj v čase. Tato analýza sloužila jako základ pro rozšíření knihovny JCMathLib verze 2.0, umožňující používat nízkoúrovňové kryptografické funkce jinak nedostupné na této platformě.
Dále proběhla analýza dostupných protiopatření vůči útokům postranními kanály na implementace provádějící skalární násobení na eliptických křivkách a jejich otestování na jednočipu ARM Cortex-M4. V reakci na výsledky této analýzy jsme vytvořili dvě optimalizované a chráněné implementace protokolu X25519. Obě implementace jsou dostupné pod permisivní otevřenou licencí (MIT).
Tato zpráva shrnuje v českém jazyce nejdůležitější zjištění, pro plný rozsah analýz doporučujeme přílohy v anglickém jazyce. Část prací úzce souvisí i s výsledky dosaženými v rámci etapy #10, pro omezení překryvu popisu nástrojů uvádíme potřebné informace na jednom místě a doporučujeme tedy přečíst i související výsledky etapy #10.

Veškeré plánované činnosti v rámci Etapy 9 byly úspěšně dokončeny a průběžné výsledky byly prezentovány aplikačnímu garantovi (NUKIB) v těchto termínech:

- **22.5.2025, Brno.** Představení pokroků v nástrojích.
- **18.9.2023 dopoledne, Praha.** Kontrolní den.
- **18.9.2023 odpoledne, Praha.** Rozšířená diskuze formou workshopu.
- **6.12.2023, Praha.** Diskuze ohledně současných výsledků, plánovaní prací pro 2024.

2 Výzkum metod protiopatření eliminující útoky postranními kanály, SW prototypová implementace vybraných primitiv na jednočipy a a čipové karty

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 oblasti:

- **CARDIS23 paper:** Přehled použití JavaCard v certifikačních dokumentech, podpora a využití kryptografických algoritmů.
- **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.
- **sca25519:** Principy návrhu ochran ECC pro křivky 25519 proti útokům postranními kanály.

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

2.1 Přehled frekvence podpory JavaCard algoritmů v softwarových implementacích s otevřeným zdrojovým kódem

JavaCard je v současné době nejrozšířenější platformou pro kryptografické čipové karty, s více než 20 miliardami vyrobených čipů. Od zveřejnění první verze JavaCard API před více než dvaceti lety již vzniklo třináct revizí specifikace, dosud ale chyběla systematická analýza použití nově přidaných funkcí, kryptografických algoritmů a jejich parametrizací i analýza celkové aktivity ekosystému. Z hlediska projektu Ai-Sectools je taková analýza přínosná z důvodu vytvoření znalosti ohledně dostupných funkcí použitelných pro bezpečnou implementaci na kryptografických čipových kartách i identifikaci dlouhodobých trendů celého ekosystému.
Identifikovanou mezeru zaplňujeme mapováním aktivity ekosystému JavaCardu z veřejně dostupných zdrojů s důrazem na:

1. Bezpečnostní certifikační dokumenty dostupné pod schématy Common Criteria a FIPS140.
2. Aktivitu a zdroje vyžadované pro JavaCard applety s otevřeným zdrojovým kódem.

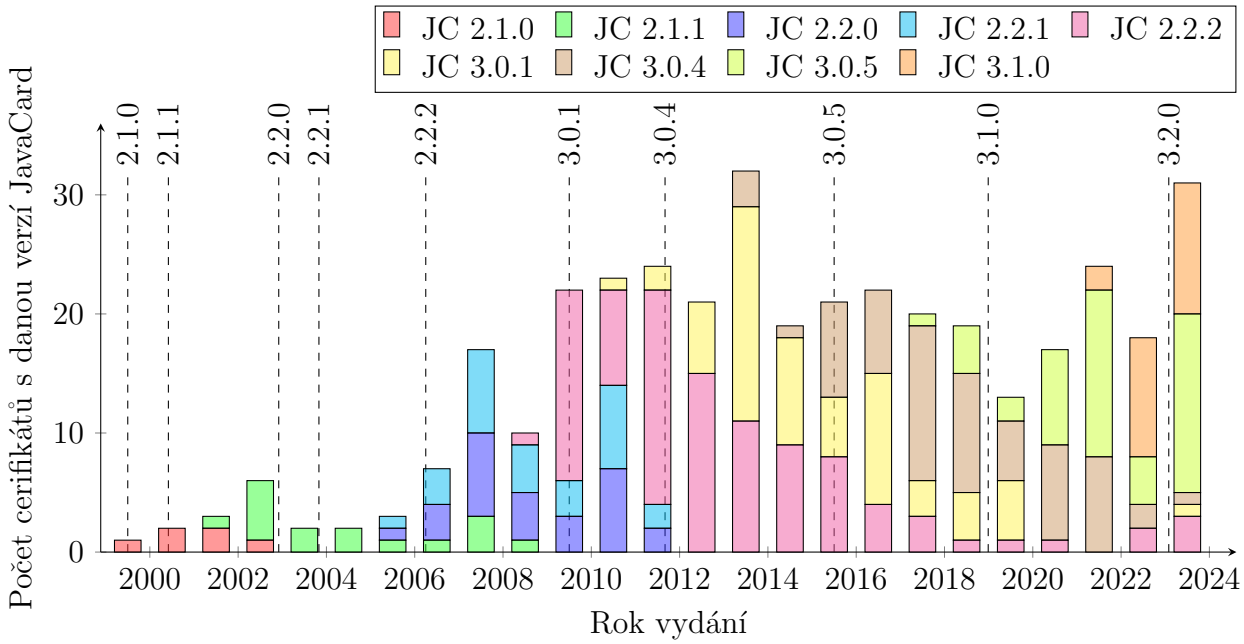
Výsledky byly prezentovány na konferenci CARDIS 2023 a celá verze článku je dostupná na `https://crocs.fi.muni.cz/papers/cardis2023`.
Analýza provedená na všech certifikátech vydaných mezi lety 1997 a 2023 a na více než 200 veřejných JavaCard appletech ukazuje, že nové funkce ze specifikace JavaCard jsou přijímány poměrně pomalu. Obvykle trvá šest i více let než dojde k široké podpoře publikovaného algoritmu v reálných čipových kartách, jak ukazuje analýza zachycená na obrázcích 1 a 2.
Applety s otevřeným kódem využívají nové funkce typicky ještě později, pravděpodobně kvůli obtížné dostupnosti nových čipových karet odběratelům pořizujícím jen malý počet kusů. Navíc téměř 70% konstant definovaných ve specifikaci JavaCard API není vůbec využito. Naopak přenositelnost appletů se s novějšími kartami zlepšuje, a požadavky na RAM paměť (cenný zdroj na čipových kartách) jsou obvykle malé.

2.1.1 JavaCard platforma v certifikovaných zařízeních

Od roku 2009 je každý rok certifikováno dvacet i více produktů s převážně vzestupnou tendencí jak je zachyceno na obrázku 1. V letošním roce 2023 (do konce října) bylo certifikováno již nejméně 30 produktů souvisejících s JavaCard platformou.

Tabulka 1: Algoritmy detekované v CC and FIPS certifikátech na základě hledání prefixu `ALG_`, které zároveň nejsou součástí oficiální verze specifikace JavaCard API.

ALG_AES_BLOCK_128_CBC_NOPAD_STANDARD	ALG_DES_CMAC	ALG_EC_SVDP_DHC_GK
ALG_AES_CBC_ISO9797_M2_STANDARD	ALG_DES_CMAC8	ALG_EC_SVDP_DHC_PACE
ALG_AES_CBC_ISO9797_STANDARD	ALG_DES_ECB_PKCS7	ALG_EC_SVDP_DH_GK
ALG_AES_CBC_PKCS7	ALG_DES_MAC128_ISO9797_1_M2_ALG3	ALG_ED25519PH_SHA_512
ALG_AES_CMAC128	ALG_DES_MAC_8_NOPAD	ALG_MONT_DH_25519
ALG_AES_CMAC16	ALG_ECDSA_RAW	ALG_RSA_SHA256_PKCS1
ALG_AES_CMAC16_STANDARD	ALG_ECDSA_SHA224	ALG_RSA_SHA256_PKCS1_PSS
ALG_AES_CMAC8	ALG_ECDSA_SHA256	ALG_RSA_SHA_1_RFC2409
ALG_AES_ECB_PKCS7	ALG_ECDSA_SHA256_LDS	ALG_RSA_SHA_256_ISO9796
ALG_AES_MAC_128_ISO9797_1_M2_ALG3	ALG_ECDSA_SHA384	ALG_SHA2_CHAIN
ALG_AES_OFB	ALG_ECDSA_SHA384_LDS	ALG_SHA_CHAIN
ALG_DES_CBC_PKCS7	ALG_ECDSA_SHA_LDS	
ALG_DES_CMAC	ALG_EC_SVDP_DHC_GK	

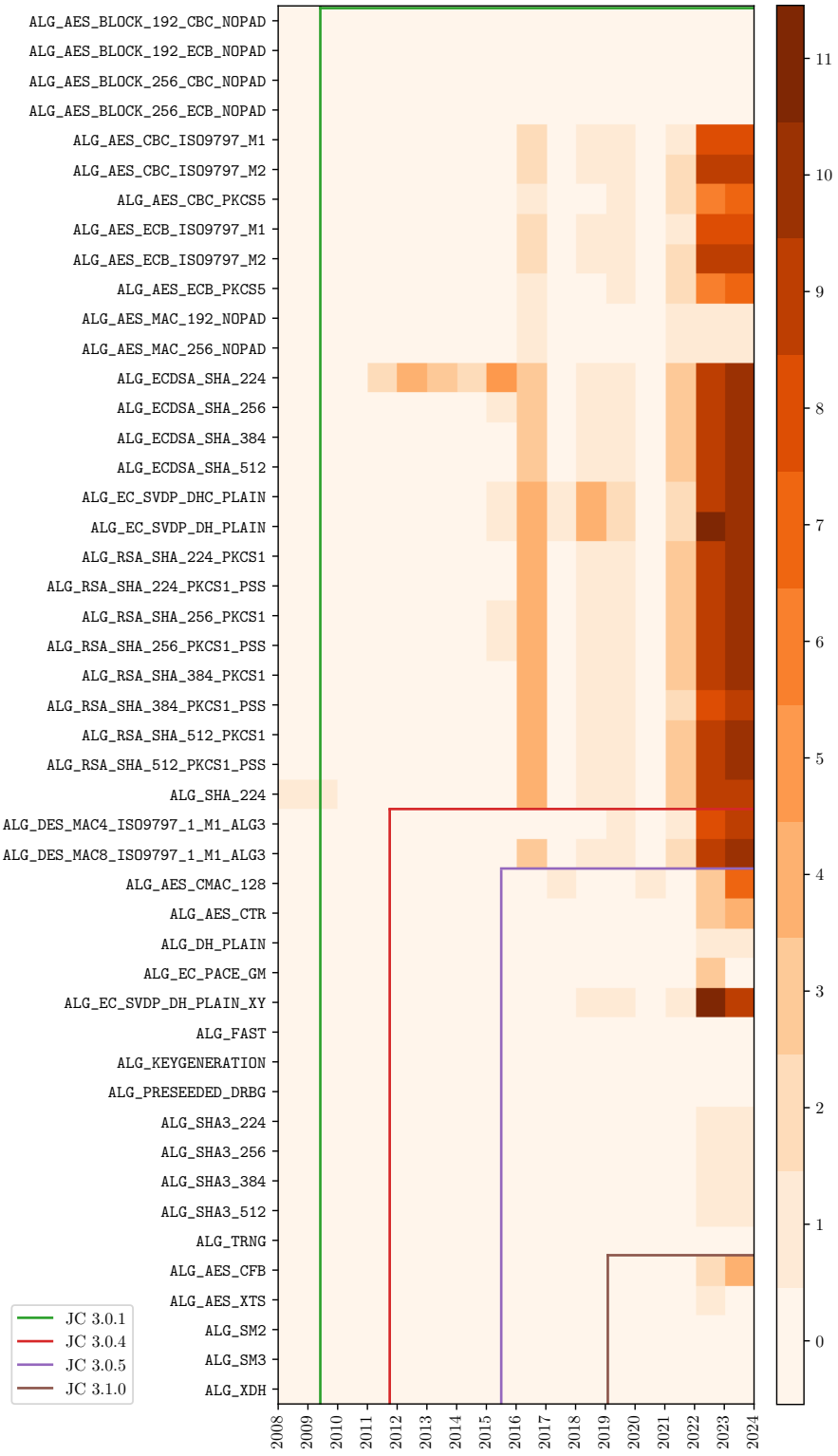


Obrázek 1: Počet certifikovaných zařízení dle Common Criteria a FIPS140 zmiňující konkrétní verzi JavaCard specifikace (rok 2023 je pouze do konce října). V případě více detekovaných verzí uvádíme v grafu pouze nejvyšší verzi.

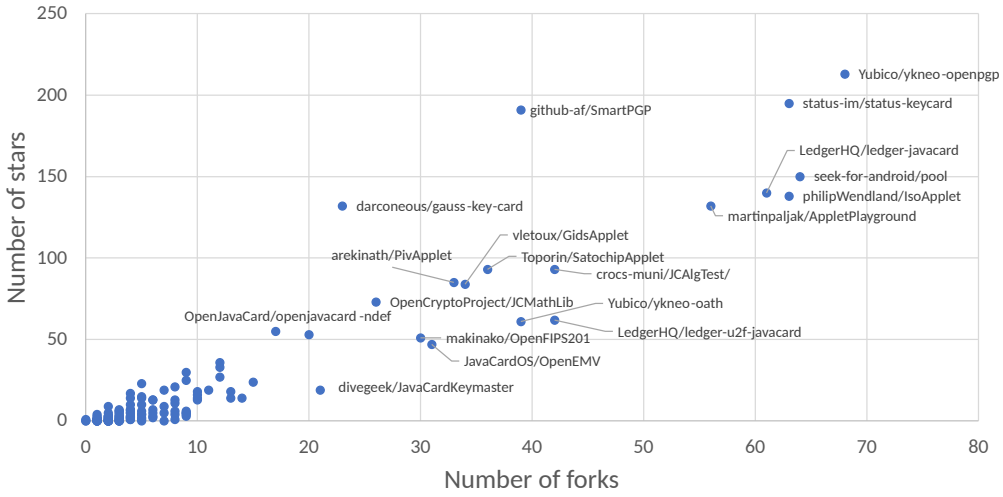
V rámci analýzy jsme také detekovali konstanty označující kryptografické algoritmy, které však nejsou uvedeny v JavaCard API uvádíme v tabulce 1. Využití těchto algoritmů vyžaduje využití proprietárních knihoven a je tedy i obtížnější provést jejich nezávislou analýzu, neboť přístup k takovým knihovnám je výrazně omezen.

2.1.2 Analýza aktivity ekosystému s otevřeným zdrojovým kódem

Pro analýzu ekosystému s otevřeným zdrojovým kódem jsme využili kombinace námi udržovaného seznamu s JavaCard applety obsahující přes 200 projektů a možností statické a dynamické analýzy nástroje JCProfilerNext (vyvíjeného v rámci projektu Ai-Sectools a popsaném detailněji v roční zprávě 2022). Nejvýznamnější projekty jsou anotovány jménem na obrázku 3.



Obrázek 2: Počet Common Criteria and FIPS140 certifikátů obsahujících zmínku o daném JavaCard algoritmu (např. ALG_EC_PACE_GM) ze specifikace API 3.0.1 v průběhu času.

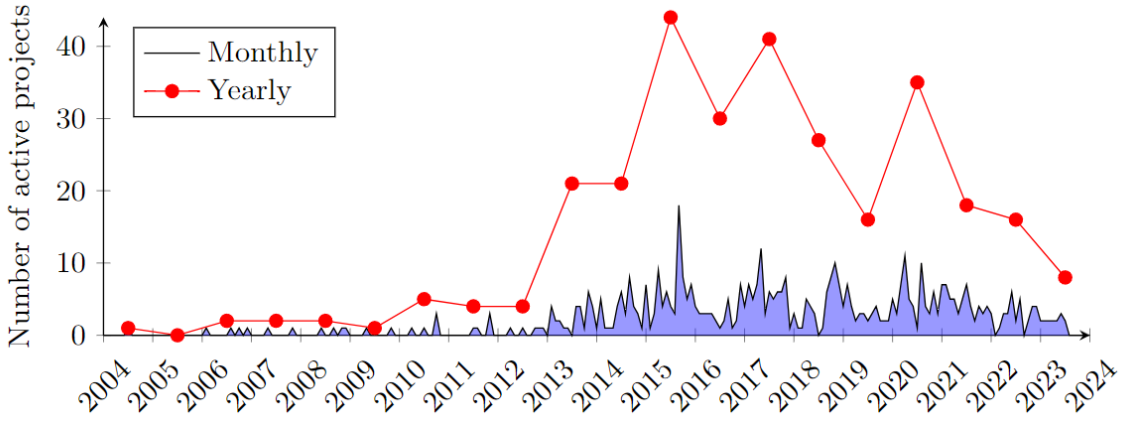


Obrázek 3: Popularita JavaCard repozitářů z hlediska počtu vývojářských kopií (fork) a popularity (stars) na platformě GitHub. Projekty s alespoň 20 kopiemi jsou anotovány jménem.

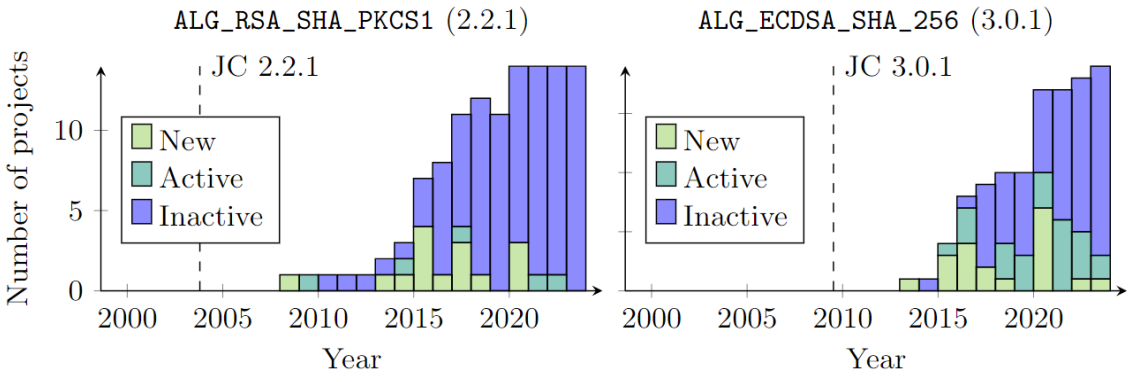
Díky využití verzovacího nástroje *git* můžeme analyzovat vývoj projektů v čase tak, že pro analýzu v daném měsící nebo roce použijeme verzi projektu z daného časového období. Následně lze provést statickou i dynamickou analýzu dané verze zdrojových kódů s postupným opakováním pro celý interval (v našem případě roky 2004 až 2023).
Statická analýza umožňuje detekovat použití všech konstant zavedených specifikací JavaCard v OSS projektech. Pro příklad uvádíme na obrázku 5 srovnání četnosti použití digitálních podpisů s algoritmem RSA versus ECDSA. Celý seznam lze nalézt v doplňujících materiálech na `https://crocs.fi.muni.cz/papers/cardis2023`.
Zatímco v případě certifikovaných zařízení jsme viděli zvyšující se množství certifikovaných zařízení s JavaCard platformou, JavaCard ekosystém s otevřeným zdrojovým kódem (OSS) se sice stal aktivnějším od roku 2013, ale v posledních dvou letech výrazně klesá, jak je zachyceno na obrázku 4. Zvýšení aktivity v roce 2013 bylo pravděpodobně způsobeno širší a volnější dostupností čipových karet v malém množství – jedním z důležitých požadavků pro zapojení vývojářů mimo velké zavedené firmy. Naopak pokles po roce 2022 může značit úbytek zájmu o tuto oblast z pohledu OSS vývojářů, potenciálně z důvodu pomalého zavádění nových algoritmů a výrazného omezení přístupu k nízkoúrovňovějším funkcím. Tato omezení často zabraňují experimentům s novými nápady – častá motivace zapojení OSS vývojářů.

2.1.3 Další informace

Detailní výsledky analýz JavaCard ekosystému jsou popsány v článku “The adoption rate of JavaCard features by certified products and open-source projects” (CARDIS 2023) [9], přílohové materiály jsou dostupné na stránce `https://crocs.fi.muni.cz/papers/cardis2023`.



Obrázek 4: Počet open-source projektů s alespoň jednou úpravou zdrojových kódů (commit) za měsíc (černá čára) resp. za rok (červená čára). Údaje pro rok 2023 jsou pouze do konce října.



Obrázek 5: Porovnání časové adopce algoritmů `ALG_RSA_SHA_PKCS1` a `ALG_ECDSA_SHA_256`. Digitální podpis ECDSA byl sice poprvé použit v OSS repozitářích až v roce 2013, zároveň je ale stále nově přidáván do zdrojového kódu aktivních projektů, zatímco podpis využívající algoritmus RSA byl poprvé začleněn v roce 2008, ale s klesající frekvencí a po roce 2020 již nebyl nově přidán ani jednou.

2.2 *JCMathLib 2.0*: matematická knihovna pro platformu JavaCard

V letošním roce jsme provedli další rozšíření knihovny JCMathLib, která se stala součástího oficiálního vydání verze 2.0 a je nyní zakomponována v upstreamovém repozitáři projektu. Součástí těchto rozšíření bylo přidání podpory nových typů karet od výrobce Infineon, sjednocení API a konvencí knihovny, a přidání několika funkčních vylepšení, která jsou detailněji popsána v následujících podsekcích.

2.2.1 Efektivnější akcelerace modulárního násobení pomocí RSA koprocesoru

Standardní otevřené JavaCard API poskytuje pouze velmi omezený přístup k hardwarovým akcelerátorům daného zařízení, jejichž použití je však nezbytné pro efektivní provádění operací používaných v kryptografických algoritmech. Knihovna JCMathLib využívá tohoto omezeného rozhraní pro reimplementaci (částečně) akcelerovaných nízkoúrovňových operací.
Původní implementace modulárního násobení v JCMathLib byla odvozena ze vzorce pro převod modulárního umocňování na násobení založeného na formuli $2ab = (a+b)^2 - a^2 - b^2$ a prováděla výpočty s modulem o bitové délce větší než $(a + b)^2$, což po vynásobení čísel vyžadovalo výpočetně náročné dělení se zbytkem (prováděné v JCVM). Tento přístup byl pravděpodobně zvolen ze dvou důvodů. Prvním důvodem byly omezení vstupů do algoritmu RSA na starších čipových kartách, které způsobily, že modulární umocňování s modulem o menší bitové délce než je minimální povolená algoritmem RSA[1] bylo neefektivní, protože každý výstup by musel být dodatečně redukován pomocí dělení se zbytkem. Druhým důvodem bylo, že při použití tohoto přístupu nikdy nedošlo k modulární redukci při žádném z umocňování, což zaručilo získání sudé hodnoty $2ab \pmod q$, kterou lze bitově posunout doprava a získat přímo ab.
Pro novou implementaci modulárního násobení jsme použili výpočet založený na vzorci $4ab = (a+b)^2 - (a-b)^2$, který vyžaduje o jedno umocňování méně než vzorec $2ab$ a zároveň pro zvýšení efektivity implementace jsme rozlišili mezi výpočtem na starších kartách, kde je omezení na minimální délku modula a novějších kartách, které jej již nemají.
Pro starší čipové karty je výhodnější použít neredukovanou variantu, která se nepotýká s problémem neefektivního dělením čtyřmi, jak uvádí Sterckx et al. [8], protože bitový posun lze provést přímo o 2 bity. Tato změna vedla k 8% zrychlení (při operacích s 256-bitovými čísly), protože náklady výpočtu jsou stále dominovány dělení zbytkem.
Nicméně, pro novější paměťové karty jsme funkci reimplementovali tak, aby výpočty probíhaly přímo modulo q, což (v závislosti na bitové délce q) zefektivní sčítání a odčítání a úplně se vyhne dodatečnému dělení se zbytkem. Navíc, s výpočtem založeném na vzorci $4ab$ a následující transformací se nám podařilo vyhnout problému při modulárním dělení čtyřmi, diskutovaném v článku od Sterckx et al. [8]:

$$c \equiv (a + b)/2 \pmod q$$
$$ab \equiv c^2 - (c - b)^2 \pmod q$$

Tímto přístupem jsme byli schopni dosáhnout více než 4,5x zrychlení oproti předchozí implementaci (při 256-bitových operacích), z čehož kolem 0,7x lze připsat použití vzorce $4ab$.

2.2.2 Změna zarovnání interní reprezentace typu BigNat

Původní interní reprezentace typu BigNat využívala pole bajtů `byte[]` fixní délky, ve kterém byla uložená hodnota zarovnaná doleva. Tento přístup se ukázal jako neefektivní, protože řada

[1] Algoritmus `ALG_RSA_NOPAD` obvykle vyžaduje modul alespoň o délce 512 nebo 1024 bitů.

operací potřebuje měnit velikost reprezentovaného čísla, což v důsledku vyžadovalo provádět kopie v rámci interního pole bajtů. Z toho důvodu jsme změnili zarovnání interní reprezentace doprava, což eliminovalo zbytečné kopie při změně velikosti reprezentovaného čísla a vedlo k celkovému zrychlení knihovny.

2.2.3 Alternativní implementace typu BigNat pro karty podporující typ `int`

Některé novější čipové karty platformy JavaCard umožňují kromě nativního 16-bitového typu `short` používat i nativní 32-bitový typ `int`. Použití 32-bitového typu by v případě, že daná čipová karta obsahuje 32-bitový procesor, mohlo umožnit až dvojnásobné zrychlení provádění některých operací. Rozhodli jsme se tedy tuto změnu implementovat.
Provedení této změny však vyžadovalo návrh, který umožní zároveň podporovat dvě implementace knihovny – jednu využívající typ `int`, který nemusí podporovat všechny karty, a druhou, která bude z důvodu zpětné kompatibility využívat pouze typ `short`. Abychom minimalizovali množství práce potřebné pro udržování dvou implementací, vyčlenili jsme ze třídy `BigNat` operace, které vyžadují přímý přístup k interní reprezentaci do nové třídy `BigNatInternal`, a zbylé operace ve třídě `BigNat` upravili tak, aby pouze používaly rozhraní třídy `BigNatInternal`. Díky těmto úpravám se minimalizovalo množství duplikace kódu mezi implementacemi. Varianta využívající typ `int` je dostupná ve větvi `ints` v repozitáři projektu.

2.2.4 Další informace

Knihovna je dostupná v repozitáři `https://github.com/OpenCryptoProject/JCMathLib`.

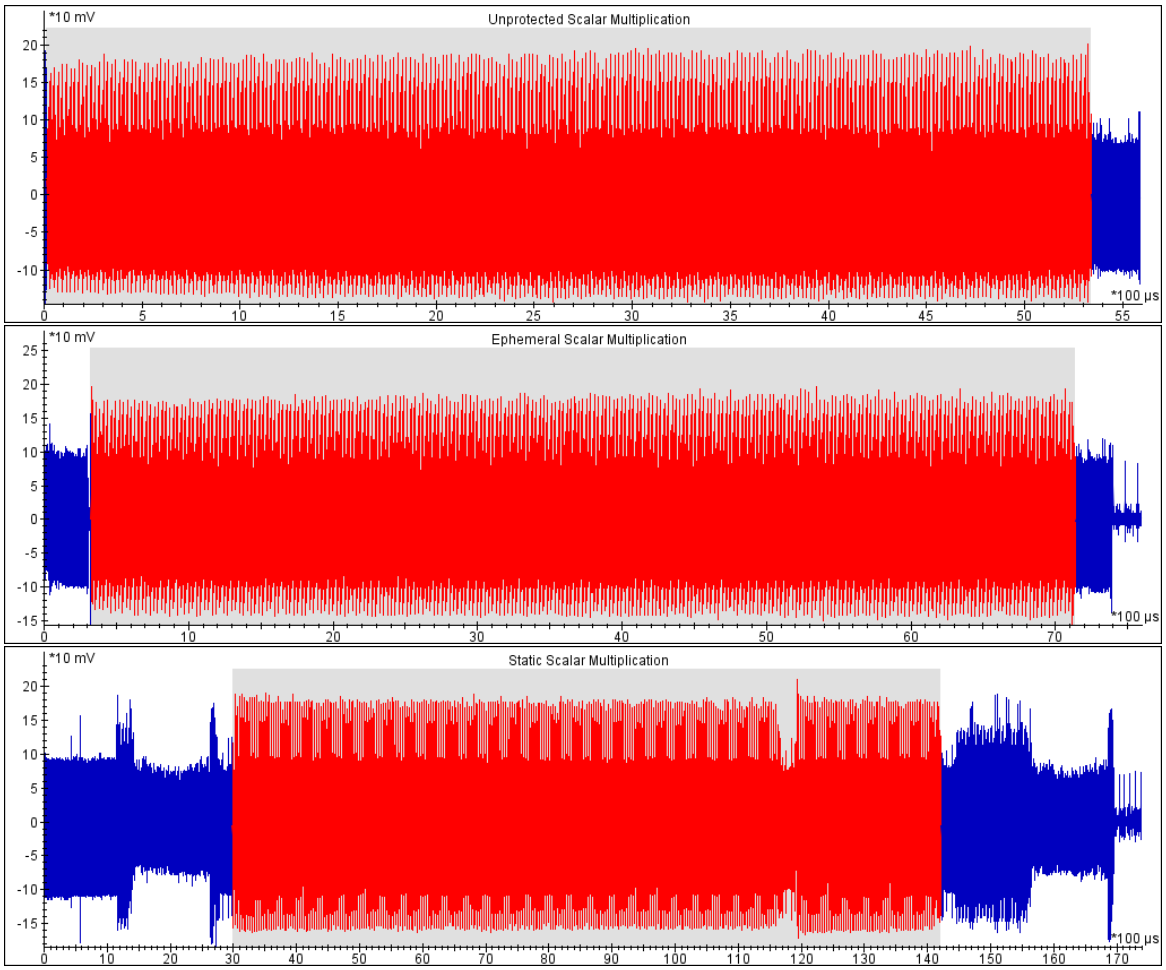
2.3 Bezpečná implementace ECC algoritmů na jednočipech MCU

V rámci projektu jsme provedli analýzu dostupných technik pro bezpečnou implementaci kryptografických algoritmů na bázi eliptických křivek a vytvořili konkrétní chráněné implementace algoritmu X25519 pro platformu ARM Cortex-M4. Detailní analýza je uvedena v práci [1], zde uvádíme krátké shrnutí.
Cílem práce je nejen přehled a systematizace současného stavu útoků postranními kanály (ang. zkratka SCA (Side-Channel Analysis) a chybových injekcí (ang. zkratka FI (Fault Induction) na ECC, ale i praktická aplikace těchto znalostí do veřejně dostupných implementací. Konkrétně představujeme dvě pečlivě optimalizované softwarové implementace Diffie-Hellmanovy výměny klíčů X25519 [3] s rozsáhlými nejmodernějšími protiopatřeními proti SCA a FI zaměřenými na populární mikroprocesor ARM Cortex-M4. Obě implementace se liší cílovým scénářem použití, úrovní poskytované ochrany a celkovým zpomalením vůči nechráněné implementaci:

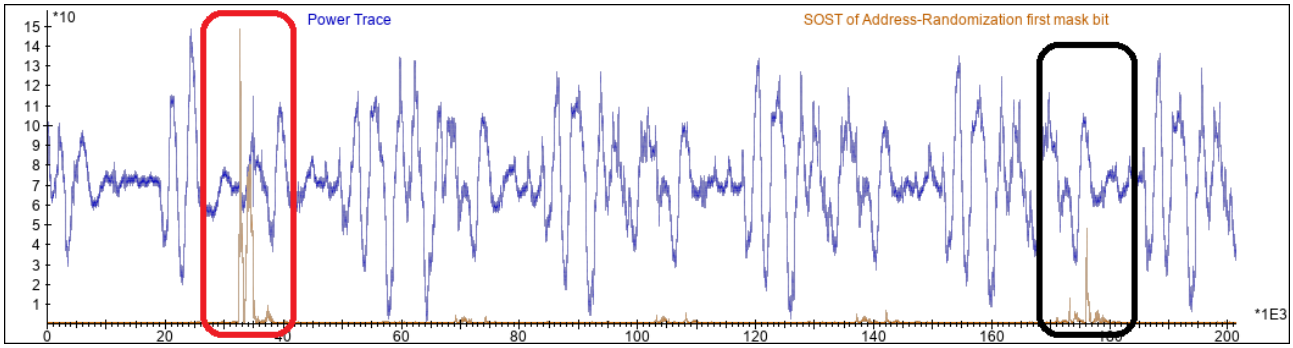
1. **Ochrana krátkodobých (efemerních) klíčů**. Implementace nabízí ochranu, o které se domníváme, že je dostatečná pro ochranu krátkodobých/jednorázových klíčů, tj. ve scénáři, kdy útočníci nejsou schopni provádět efektivně diferenciální SCA nebo FI útoky z důvodu velmi krátkého používání napadaného klíče (např. pouze jediná operace).

2. **Ochrana dlouhodobějších (statických) klíčů**. Implementace se zaměřuje na ochranu X25519 se statickými klíči, čímž přidává ochranu proti diferenciálním útokům které může útočník využít při opakovaném sledování operací se stejným (statickým) klíčem.

Přidaná komplexita naší implementace ve srovnání s pečlivě optimalizovanou základní nechráněnou implementací s ohledem na cykly procesoru je přibližně 37% pro efemerní případ a 239% pro statický případ. V absolutních číslech naše rozsáhleji chráněná implementace zaměřená na statický případ dokonce překonává implementace ECC v široce rozšířených kryptografických knihovnách (o 1% rychlejší než bearSSL a BoringSSL, 2.75x rychlejší než ARM Mbed TLS). Důvodem výrazného výkonu našich implementací je pečlivá optimalizace na úrovni assembleru. Identifikovali jsme synergie mezi SCA ochranami a výkonem například pomocí pečlivého ladění alokace registrů s cílem minimalizovat přístup do paměti.
Abychom ověřili, do jaké míry jsou naše ochrany bezpečné, provedli jsme experimentální evaluaci bezpečnosti. Nejprve jsme použili běžně používaný testovací vektor hodnocení úniku (angl. zkratka TVLA (Test Vector Leakage Assesment)) [2] v různých nastaveních pro detekci různých typů úniků (výsledky zobrazeny na obrázku 6). Dále jsme provedli profilované útoky s jednou stopou, abychom potvrdili, že naše protiopatření poskytují určitou úroveň ochrany i proti těmto pokročilejším útokům postranními kanály.
Zejména pro profilované útoky proti statické implementaci jsme vypočítali body zájmu (angl. zkratka POI (Point Of Interest)) pro různé masky. Například obrázek 7 představuje graf detekce POI pro nejvýznamnější bit masky randomizace adresy. První oblast označená červeně odpovídá generování masky a druhá černá část iterace skalárního násobení. Vzhledem k tomu, že obě oblasti obsahují vysoké vrcholy sum-of-squared t-values (SOST, tj. 150 a 40, v tomto pořadí), útok single-trace je relativně úspěšný 63-64%, ale nestačí k obnovení masky. Ostatní profilované útoky byly ještě méně úspěšné.
Je překvapivé, že navzdory rozsáhlé literatuře o útocích proti implementacím ECC nám není známa jediná práce popisující veřejně dostupnou implementaci s podobnou úrovní ochrany proti SCA a FI. Existují implementace ECC, které *tvrdí*, že nabízejí rozsáhlou ochranu. Například společnost NXP inzeruje *"bezpečný řadič čipových karet"* SmartMX2-P40 podporující *"kryptografii DES, AES, ECC, RSA, výpočet hashe a generování náhodných čísel"* a tvrdí, že ochranné



Obrázek 6: Odběrové křivky nechráněné implementace (nahoře), implementace pro ochranu krátkodobých (uprostřed) a statických (dole) klíčů. Červená část křivky obsahuje skalární násobení, modrá část dodatečný kód implementovaných ochran. V případě ochrany statických klíčů je navíc vidět i dodatečná ochrana (64bit scalar splitting) následující v červené oblasti za základním skalárním násobením.



Obrázek 7: Odběrová křivka operace (modrá křivka) a odpovídající součet t-hodnot (hnědá křivka, sum-of-squared t-values, SOST) pro randomizaci adresy demonstrující existenci úniku informace postranním kanálem potenciálně využitelnou pro profilující útok v jediné stopě. Tento útok je ale úspěšný pouze v cca 63-64% případech a nestačí na obnovení masky.

mechanismy *"neutralizují všechny útoky postranními kanály a chybové injekce, stejně jako snahy o reverzní inženýrství"* [6]. Stejně jako u všech komerčně dostupných čipových karet jsou však všechny podrobnosti o implementaci utajovány, což omezuje akademickou diskusi a veřejnému hodnocení těchto tvrzení. Výsledkem této situace je, že řada článků prezentuje útoky postranními kanály proti implementacím ECC, které nikdy netvrdily, že jsou proti takovým útokům chráněny; viz například [5, 7].
Tím nechceme říci, že neexistují žádné veřejné implementace ECC obsahující *některá* protiopatření proti *některým* konkrétním útokům. Naopak, většina prací popisujících útoky také navrhuje odpovídající protiopatření a některé z těchto prací také představují i praktickou implementace těchto protiopatření. Zároveň ale platí, že více než tři desetiletí od zavedení ECC a dvě desetiletí po objevení útoků postranními kanály (SCA) však stále neznáme uspokojivý odhad nákladů na implementace ECC s *úplnými*, *specifickými* a *aditivními* SCA protiopatřeními (viz [4, Sec. 6.2]), ani zda jsou takové implementace vůbec dosažitelné s ohledem na cenu velikosti, paměť, výkonu a energie, které jsou pro levná zařízení typické. Navržená a prakticky implementovaná ochrana pro statické klíče chráněná proti klasickým útokům postranními kanály i proti vložení jedné chyby na platformě ARM Cortex-M4 v článku [1] ukazuje zvýšení dobu výpočtu o cca 239%.

Dostupnost softwaru a další informace

Přehled existujících ochran a návrh specifických ochran pro diskutované scénáře je popsán v článku "SoK: SCA-secure ECC in software – mission impossible?" (CHES 2023) [1], zdrojový kód je dostupný v repozitáři
`https://github.com/sca-secure-library-sca25519/sca25519`.

3 Závěr

Veškeré plánované činnosti v rámci Etapy 9 byly úspěšně dokončeny včetně prezentace průběžných výsledků aplikačnímu garantovi během tří osobních setkání a workshopů (květen, září a prosinec 2023).

V rámci Etapy 9 jsme vytvořili nástroje potřebné pro zmapování dostupných algoritmů a funkcí na kryptografických čipových kartách s platformou JavaCard v certifikovaných zařízeních a jejich využití v implementacích s otevřeným zdrojovým kódem. Dále jsme vytvořili rozšíření knihovny JCMathLib demonstrující možnost implementace nízkoúrovňovejších operací na platformě JavaCard díky kreativnímu využití existujících hardwarově akcelerovaných operací. Třetí pokrytou oblastí byl návrh a praktická implementace zabezpečené verze algoritmu X25519 pro platformu Arm Cortex-M4 včetně ověření odolnosti vůči útoku postranními kanály a chybovým útokům. Tuto implementaci lze využít pro zabezpečení dalších algoritmů ECC využívajících skalární násobení.

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] Batina, Lejla, Chmielewski, Łukasz, Haase, Björn, Samwel, Niels, and Schwabe, Peter. *SoK: SCA-secure ECC in software – mission impossible?* IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023(1):557–589, 2022. doi:10.46586/tches.v2023.i1.557-589.
URL `https://tches.iacr.org/index.php/TCHES/article/view/9962`

[2] Becker, Georg T., Cooper, Jeremy, DeMulder, Elke, Goodwill, Gilbert, Jaffe, Joshua, Kenworthy, Gary, Kouzminov, Timofei, Leiserson, Andrew J., Marson, Mark E., Rohatgi, Pankaj, and Saab, Sami. *Test vector leakage assessment (TVLA) methodology in practice.* International Cryptographic Module Conference, 2013.

[3] Bernstein, Daniel J. *Curve25519: new Diffie-Hellman speed records.* In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, Public Key Cryptography – PKC 2006, vol. 3958 of *LNCS*, pp. 207–228. Springer, 2006. `http://cr.yp.to/papers.html\#curve25519`.

[4] Fan, Junfeng and Verbauwhede, Ingrid. *An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost.* In David Naccache, editor, Cryptography and Security: From Theory to Applications, vol. 6805 of *LNCS*, pp. 265–282. Springer, 2012.

[5] Genkin, Daniel, Pachmanov, Lev, Pipman, Itamar, and Tromer, Eran. *ECDH Key-Extraction via Low-Bandwidth Electromagnetic Attacks on PCs*, 2016. `https://web.eecs.umich.edu/~genkin/papers/ecdh.pdf`.

[6] NXP. *SmartMX2 P40 family P40C012/040/072 Secure smart card controller*, 2015. `https://www.nxp.com/docs/en/data-sheet/P40C040_C072_SMX2_FAM_SDS.pdf`.

[7] Samwel, Niels, Batina, Lejla, Bertoni, Guido, Daemen, Joan, and Susella, Ruggero. *Breaking Ed25519 in WolfSSL.* In Nigel P. Smart, editor, Topics in Cryptology – CT-RSA 2018, vol. 10808 of *LNCS*, pp. 1–20. Springer, 2018. `https://eprint.iacr.org/2017/985`.

[8] Sterckx, Michaël, Gierlichs, Benedikt, Preneel, Bart, and Verbauwhede, Ingrid. *Efficient implementation of anonymous credentials on Java Card smart cards.* In 2009 First IEEE International Workshop on Information Forensics and Security (WIFS), pp. 106–110. IEEE, 2009.

[9] Zaoral, Lukas, Dufka, Antonin, and Svenda, Petr. *The adoption rate of JavaCard features by certified products and open-source projects.* In Proceedings of the 22nd Smart Card Research and Advanced Application Conference. Springer, 2023.

Příloha: Analýza rizik

Tabulka 2: Analýza rizik relevantních k Etapě 9.

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
Nedostatečná rychlost vykonávání operace skalárního násobení pro dané MCU	Nemožnost využít navrženou implementaci na daném MCU	Praktické snížení použitelnosti implementace	-	Optimalizace implementace pro dané MCU