# Investigação do Ataque Flush+Reload via Web para Extração de Chaves RSA em Alvos Nativos

**Felipe S. Simões**[1], **Lucila M. S. Bento**[2], **Raphael C. S. Machado**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Av. Gal. Milton Tavares de Souza, s/n - São Domingos, Niterói/RJ, 24210-310

[2]Instituto de Matemática e Estatística – Universidade do Estado do Rio de Janeiro (UERJ)
R. São Francisco Xavier, 524 - 6o. andar - Maracanã, Rio de Janeiro/RJ, 20550-000

felipe_simoes@id.uff.br, lucila.bento@ime.uerj.br, raphaelmachado@ic.uff.br

***Abstract.*** *WebAssembly enables the execution of complex applications in the browser, raising the question about the potential adaptation of sophisticated native attacks, such as Flush+Reload—a high-resolution cache side-channel attack. This work experimentally investigates the feasibility of executing Flush+Reload from the browser against an external native GPG process to extract RSA keys, comparing this approach with native implementations. Results indicate a clear contrast: while native versions achieved partial success (nearly 50% accuracy), the web-based approach proved ineffective (accuracy below 4%), limited by inherent challenges in timing resolution and noise within the browser environment. These findings indicate a fundamental difference in the current exploitability of this attack vector against external targets when comparing web and native environments.*

***Resumo.*** *O WebAssembly viabiliza a execução de aplicações complexas no navegador, levantando a questão sobre a potencial adaptação de ataques nativos sofisticados, como o Flush+Reload — um ataque de canal lateral de cache com alta resolução temporal. Este trabalho investiga experimentalmente a viabilidade de executar o Flush+Reload a partir do navegador contra um processo GPG nativo externo para extrair chaves RSA, comparando esta abordagem com implementações nativas. Os resultados indicam um contraste claro: enquanto as versões nativas alcançaram sucesso parcial (quase 50% de acerto), a versão web se mostrou ineficaz (com taxa de acerto inferior a 4%), limitada por questões inerentes de resolução temporal e ruído no ambiente do navegador. Desta forma, os resultados indicam uma diferença fundamental na capacidade atual de exploração deste vetor de ataque entre os ambientes web e nativo.*

## 1. Introdução

Os navegadores web são atualmente uma das principais aplicações usadas pelas pessoas, executando desde tarefas simples, como a navegação em páginas web ou leitura de e-mails, até aplicações complexas que antes demandavam software nativo, como a edição colaborativa de documentos, edição de imagens e jogos 3D. A evolução de tecnologias como JavaScript e, mais recentemente, WebAssembly, ampliou drasticamente as capacidades do ambiente web. Em particular, o WebAssembly promete desempenho próximo ao

nativo, abrindo a possibilidade teórica de adaptar ataques cibernéticos sofisticados, antes restritos ao ambiente nativo, para serem executados a partir de uma simples página web.

Dentre as ameaças que se beneficiam desse cenário estão os ataques de canal lateral (*side-channel attacks* — SCAs). Esses ataques exploram vazamentos não intencionais de informação decorrentes da própria execução de hardware ou software, de modo que se tornam alternativas para condução de ataques por seu potencial de burlar mecanismos de isolamento. Ataques de canal lateral baseados em cache são particularmente poderosos, pois exploram o tempo de acesso a posições de memória compartilhadas na hierarquia de cache da CPU para inferir dados secretos. O ataque Flush+Reload [Yarom and Falkner 2014] merece destaque por sua alta resolução temporal e baixo ruído, sendo capaz de, por exemplo, extrair chaves criptográficas RSA completas monitorando o acesso a endereços de memória específicos durante operações de decriptografia em implementações vulneráveis como a do GnuPG (GPG).

Originalmente demonstrado em ambiente nativo, onde o atacante possui controle direto sobre instruções privilegiadas (como `clflush` para limpar linhas de cache) e temporizadores de alta precisão (como `rdtscp`), surge a questão: seria viável adaptar um ataque de alta resolução como o Flush+Reload para o ambiente web, utilizando WebAssembly e JavaScript, para atacar um processo nativo? Essa adaptação enfrenta desafios significativos, como a ausência de instruções diretas para manipulação da cache e a necessidade de contornar as limitações e o ruído introduzidos pelos temporizadores e pelo próprio ambiente de execução do navegador. A capacidade de realizar tal ataque *cross-environment* (web para nativo) representaria uma ameaça severa, pois um usuário poderia ter chaves de aplicações nativas comprometidas apenas ao visitar uma página maliciosa.

Este trabalho investiga empiricamente a viabilidade dessa adaptação. Implementamos[1] e comparamos três versões do ataque Flush+Reload contra a implementação CRT-RSA vulnerável do GPG 1.4.13: (i) uma versão nativa utilizando a instrução `clflush`; (ii) uma versão nativa utilizando *eviction sets* [Vila et al. 2018] como alternativa ao `clflush`; e (iii) uma versão web combinando JavaScript e WebAssembly, utilizando *eviction sets* e temporizadores avançados baseados em memória compartilhada (como o *eagle timer* [Mazaheri et al. 2022]. A taxa de sucesso na recuperação de bits da chave RSA em diferentes hardwares foi avaliada. A principal contribuição deste trabalho é fornecer uma análise prática sobre os limites atuais para a execução de ataques de canal lateral de cache com alta resolução temporal a partir do navegador contra alvos nativos, mesmo com o uso de WebAssembly.

O restante deste artigo está organizado conforme descrito a seguir. A Seção 2 apresenta a fundamentação teórica necessária. A Seção 3 detalha a metodologia experimental e as implementações do ataque. A Seção 4 apresenta e analisa os resultados obtidos. A Seção 5 discute o impacto dos resultados e limitações. Finalmente, a Seção 6 conclui o trabalho e apresenta possibilidade de pesquisas futuras.

## 2. Fundamentação Teórica

Esta seção apresenta os conceitos básicos necessários para a compreensão do ataque Flush+Reload, a vulnerabilidade alvo neste artigo e os desafios de sua adaptação para o ambiente web.

[1] `https://github.com/felipeasimos/flush-reload-web`

### 2.1. Ataques de Canal Lateral via Cache LLC e Flush+Reload

Sistemas computacionais modernos utilizam uma hierarquia de caches para acelerar o acesso à memória [Tanenbaum and Bos 2014]. A *Last Level Cache* (LLC, geralmente L3) é tipicamente compartilhada entre todos os núcleos (cores) de um processador, o que, embora seja benéfico para o desempenho, cria um canal lateral: um processo malicioso pode inferir informações sobre outros processos (vítimas) observando as alterações no estado da LLC. A principal forma de exploração adotada nesse tipo de ataque é medir o tempo de acesso a um endereço de memória: um acesso rápido (*cache hit*) indica que o dado estava na cache, enquanto um acesso lento (*cache miss*) indica que o dado precisou ser buscado na memória principal.

O ataque Flush+Reload [Yarom and Falkner 2014] explora esse princípio com alta resolução temporal — ou seja, monitora acessos à memória em intervalos de tempo muito curtos, da ordem de nanossegundos ou poucos ciclos de CPU —, focando em endereços de memória compartilhados entre atacante e vítima. Tal compartilhamento de memória ocorre frequentemente com código de bibliotecas ou, como explorado neste trabalho, através da deduplicação de memória, onde o sistema operacional mapeia cópias idênticas de páginas de memória (como as do código de um executável carregado por múltiplos processos) para a mesma página física. Conforme ilustrado na Figura 1, o ataque consiste em três passos repetidos sucessivamente e em intervalos curtos de tempo:

- **Flush:** O atacante remove um bloco de memória específico (contendo o endereço monitorado) da cache LLC. Em ambiente nativo, essa ação é feita eficientemente com instruções como `clflush` em processadores Intel.
- **Wait:** O atacante aguarda um curto período, durante o qual a vítima pode (ou não) acessar o endereço monitorado, trazendo-o de volta para a LLC.
- **Reload:** O atacante acessa o mesmo endereço e mede o tempo. Um *cache hit* indica que a vítima acessou o endereço durante a espera; um *cache miss* indica que não.

Ao monitorar endereços correspondentes a instruções específicas de um algoritmo, o atacante pode inferir o fluxo de execução da vítima e, consequentemente, dados secretos.

### 2.2. Vulnerabilidade na Exponenciação no CRT-RSA do GnuPG

Por ser um padrão utilizado amplamente para diversas aplicações criptográficas [NIS 2023], o algoritmo RSA é frequentemente alvo de otimizações. Para otimizar a operação de decriptografia, implementações como a do GnuPG (GPG) em versões anteriores à 1.4.14 utilizam o Teorema Chinês dos Restos (CRT-RSA). Este método envolve exponenciações modulares que, por sua vez, são frequentemente implementadas pelo algoritmo de exponenciação por quadrados e multiplicações (*square-and-multiply*) [Gordon 1998]. O Flush+Reload explora o padrão de execução desse algoritmo para inferir dados secretos: para cada bit da chave secreta (expoente), ele realiza uma diferente ordem de operações; se o bit for '1', a sequência será quadrado, módulo, multiplicação e módulo novamente, mas se o bit for '0', a sequência será apenas quadrado, módulo e não ocorrerá multiplicação em seguida (um nova operação de módulo ou quadrado será executada). Portanto, ao monitorar os endereços das instruções de quadrado (`mpih_sqr_n` no GPG) e multiplicação (`mul_n` no GPG) com o Flush+Reload, é possível distinguir a sequência de operações e reconstruir os bits da chave privada.

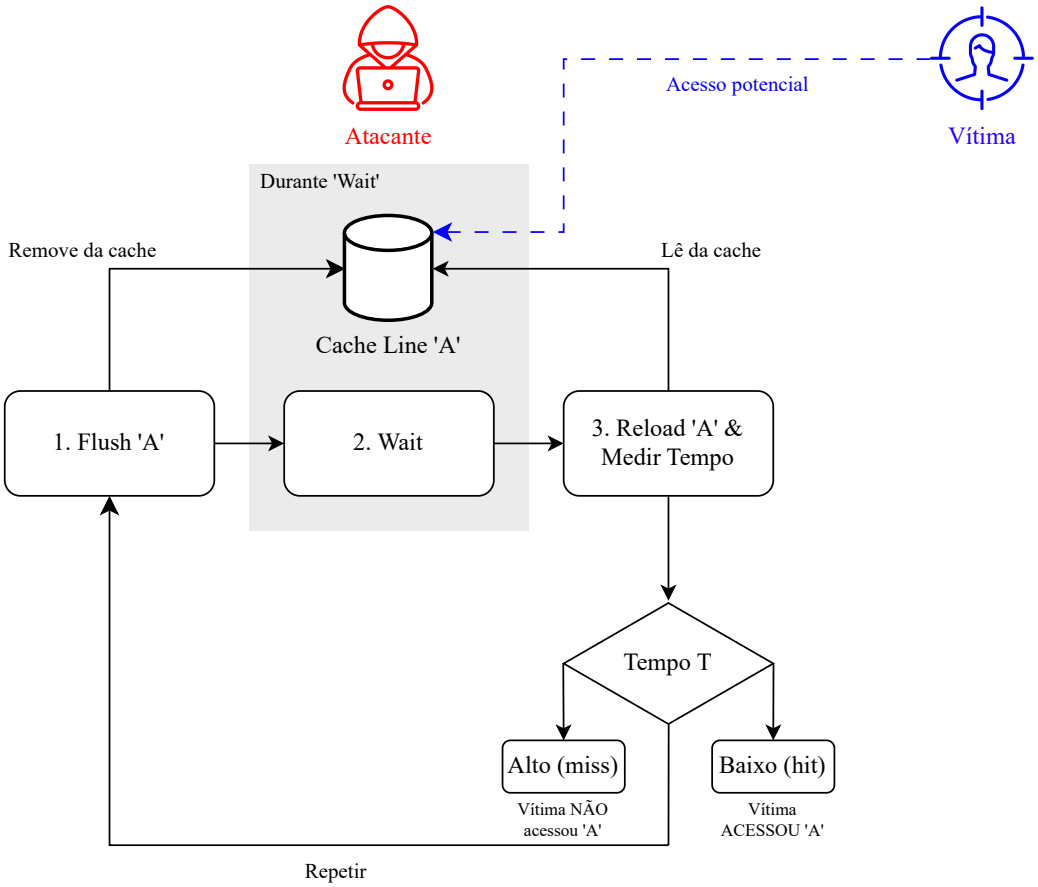



**Figure 1. Ciclo do Flush+Reload .**

### 2.3. Adaptação para o Ambiente Web

Executar o Flush+Reload a partir de um navegador web contra um alvo nativo introduz desafios significativos, pois o atacante não tem acesso direto a instruções privilegiadas como `clflush` ou temporizadores de hardware de alta precisão como `rdtscp`. Desta forma, as principais adaptações e tecnologias necessárias para a adaptação do ataque são:

- Esvaziamento da cache sem `clflush`: A técnica de *eviction sets* é a alternativa mais comum e consiste em encontrar um conjunto de endereços virtuais que, devido ao mapeamento da cache (baseado no *Set Index* e *slicing* da LLC), competem pelo mesmo conjunto de cache que o endereço alvo. Então, acessar todos os endereços do *eviction set* força a remoção (evicção) do endereço alvo da cache, simulando o efeito do `clflush`, embora geralmente de forma mais lenta e ruidosa. Um ponto importante de se observar é que encontrar *eviction sets* mínimos e eficientes é um desafio [Vila et al. 2018].
- Medição de tempo precisa: A API padrão `performance.now()` do JavaScript teve sua precisão reduzida em navegadores modernos como mitigação contra ataques temporais [Gierlichs et al. 2017]. Para obter a resolução necessária para distinguir *cache hits* de *misses* (na ordem de dezenas ou centenas de ciclos de CPU), podem ser exploradas técnicas alternativas. Contadores baseados em memória compartilhada (*SharedArrayBuffer*) e threads (*Workers* ou *WebAssembly threads*) surgiram como solução, onde uma thread incrementa um contador

em memória compartilhada em alta velocidade, e a thread atacante lê esse contador para medir intervalos de tempo curtos [Gierlichs et al. 2017]. Neste trabalho, utilizamos o *eagle timer*, uma implementação dessa ideia otimizada com WebAssembly [Mazaheri et al. 2022].
- Execução controlada em WebAssembly: O WebAssembly é essencial para implementar o loop de monitoramento e acesso à memória de forma performática. O SharedArrayBuffer permite alocar a memória necessária para a cópia do executável alvo (explorando a deduplicação) e para os *eviction sets*. Já a API Atomics do WebAssembly é crucial para garantir a ordem correta das operações de medição de tempo e acesso à memória, prevenindo reordenamentos pelo compilador ou processador que invalidariam as medições de tempo.

Embora trabalhos anteriores tenham demonstrado outros ataques de cache via web [Oren et al. 2015, Gierlichs et al. 2017, Genkin et al. 2018], a viabilidade de ataques de alta resolução temporal, como o Flush+Reload, lançados do navegador contra processos nativos externos, até onde sabemos, permanecia uma questão em aberto e é explorada neste artigo.

## 3. Metodologia

A metodologia adotada para avaliar a viabilidade da adaptação do ataque Flush+Reload para o ambiente web, incluindo o ambiente de teste, as configurações experimentais comparadas e os procedimentos de análise, é descrita a seguir.

### 3.1. Abordagem

Como o principal objetivo deste trabalho foi investigar se um ataque de canal lateral de cache com alta resolução temporal (Flush+Reload) pode ser executado com sucesso a partir de um navegador web (utilizando JavaScript e WebAssembly) contra um processo nativo externo, implementamos e comparamos três configurações distintas do ataque:

- Nativo com `clflush`: Execução em ambiente nativo (com Assembly), utilizando a instrução `clflush` para esvaziar a cache.
- Nativo com *eviction sets* : Execução em ambiente nativo (com Assembly), utilizando *eviction sets* para esvaziar a cache.
- Web com *eviction sets* : Execução a partir do navegador (JavaScript/WebAssembly), utilizando *eviction sets* e temporizadores customizados.

### 3.2. Ambiente e alvo

Os experimentos foram conduzidos em duas máquinas distintas para avaliar a influência do processador na execução do ataque: uma com Intel Core i7-6500U (LLC 4 MiB, 12-way associativity); e outra com Intel Core i5-7500T (LLC 6 MiB, 16-way associativity). Ambos utilizam páginas de memória de 4KB.

Para garantir a reprodutibilidade, utilizamos um container Docker baseado no Debian 12.5. O software alvo foi o GnuPG (GPG) versão 1.4.13, compilado com GCC 12.2 usando as flags `-g -O2`. Antes de cada execução do ataque, um novo par de chaves RSA de 2048 bits era gerado aleatoriamente.

Como alvo, consideramos a vulnerabilidade na implementação do CRT-RSA do GPG anterior a versão 1.4.13, onde o padrão de execução das operações de quadrado e multiplicação revela bits da chave privada, conforme detalhado na Seção 2.2. A capacidade de monitorar o GPG a partir de outro processo (nativo ou web) depende da deduplicação de memória realizada pelo sistema operacional, pois este mecanismo faz com que o código do GPG na memória do atacante e da vítima compartilhem as mesmas páginas físicas e, consequentemente, as mesmas linhas na cache LLC, viabilizando a observação dos acessos da vítima.

### 3.3. Implementações do Ataque

As três configurações compartilham o mesmo fluxo geral: copiar executável alvo, preparar esvaziamento, loop de monitoramento, análise. No entanto, estas diferem nas técnicas empregadas para tarefas essenciais, que incluem medição de tempo, esvaziamento da cache, garantia de ordem de execução, gerenciamento de memória e seleção de endereços-alvo.

No que se refere a medição de tempo, no ambiente nativo foi utilizada a instrução `rdtscp`, que lê o contador de ciclos do processador com baixa latência e previne reordenamento de instruções, fornecendo medições de alta precisão. Enquanto na web foi empregado o eagle timer [Mazaheri et al. 2022], um contador de alta resolução implementado em WebAssembly usando duas threads e memória compartilhada (SharedArrayBuffer), como alternativa à API `performance.now()` de precisão reduzida.

Quanto ao esvaziamento da cache (Flush/Eviction), foi realizada uma implementação em ambiente nativo do ataque usando a instrução assembly `clflush`, que remove eficientemente uma linha específica da cache LLC, e *eviction sets*, além de duas implementações (uma em ambiente nativo e outra web) usando *eviction sets*. Para cada endereço alvo, um conjunto de endereços virtuais que mapeiam para o mesmo conjunto na cache LLC foi pré-calculado. O acesso sequencial a esses endereços força a remoção do alvo. A geração seguiu abordagens baseadas em alocação de grandes buffers e algoritmos de redução [Vila et al. 2018], adaptados para cada ambiente.

Em relação à garantia de ordem de execução, no ambiente nativo, a instrução `rdtscp` possui uma barreira de serialização implícita. Adicionalmente, a instrução `mfence` foi usada para garantir que operações de memória não fossem reordenadas pelo processador ou compilador em pontos críticos. Já no ambiente web foi utilizada a API Atomics do WebAssembly (atomic.fence, i32.atomic.load) em conjunto com a criação manual de dependências de dados entre as operações de leitura do timer e acesso à memória monitorada, para mitigar o reordenamento de instruções no ambiente WebAssembly.

O gerenciamento de memória no ataque nativo utilizou alocação padrão de memória em C (`malloc`). Já o web usou SharedArrayBuffer para alocar a memória necessária ao eagle timer, à cópia do executável GPG (explorando deduplicação) e aos candidatos para os *eviction sets* . É importante destacar que a execução do código web requer que o servidor envie cabeçalhos HTTP específicos (`Cross-Origin-Opener-Policy: same-origin`, `Cross-Origin-Embedder-Policy: require-corp`) para habilitar o SharedArrayBuffer.

Para a seleção dos endereços-alvo, os endereços das instruções dentro das funções `mpih_sqr_n` (quadrado), `mpihelp_divrem` (módulo/redução) e `mul_n` (multiplicação) do GPG foram identificados usando `objdump` e selecionados seguindo recomendações para evitar ruídos de execução especulativa [Yarom and Falkner 2014], como escolher instruções fora do primeiro bloco de cache da função ou dentro de laços.

### 3.4. Sobre a execução dos experimentos e métricas

Para cada uma das 3 configurações em cada uma das 2 máquinas, o experimento foi executado 100 vezes. Em cada execução:

1. Um processo GPG iniciava a decriptografia de um arquivo usando a chave RSA gerada.
2. O processo atacante (nativo ou web) iniciava o loop de monitoramento dos endereços alvo, registrando o tempo de acesso (medido em ciclos de CPU ou ticks[2] do eagle timer) para cada um.
3. Após um número pré-definido de amostras, os tempos de acesso eram salvos.
4. Os dados de tempo eram processados: um limiar (threshold), determinado manualmente por análise visual de execuções preliminares para cada configuração, era usado para classificar cada acesso como hit ou miss.
5. A sequência resultante de hits/misses para as operações de quadrado e multiplicação era então traduzida em uma sequência de bits da chave usando uma máquina de estados baseada em trabalhos anteriores [da Silva Simões et al. 2021, Ge et al. 2015].
6. A métrica de avaliação foi o percentual de acerto: a porcentagem de bits recuperados pelo ataque que correspondiam corretamente aos bits da chave secreta original na mesma posição. A média e a variância desse percentual ao longo das 100 execuções foram calculadas.

## 4. Resultados

Esta seção apresenta os resultados obtidos a partir da execução das três configurações do ataque Flush+Reload (nativo com `clflush`, nativo com *eviction setS* , Web com *eviction sets*) nas duas plataformas de hardware (Intel i7-6500U e i5-7500T), conforme descrito na Seção 3. A análise consiste na classificação dos tempos de acesso em *cache hits* ou *cache misses* utilizando limiares (*thresholds*) definidos manualmente (Tabela 1) e na subsequente tradução da sequência de operações detectadas em bits da chave RSA por meio de uma máquina de estados [da Silva Simões et al. 2021]. A métrica principal é o percentual médio de acerto dos bits recuperados em 100 execuções para cada cenário, juntamente com sua variância.

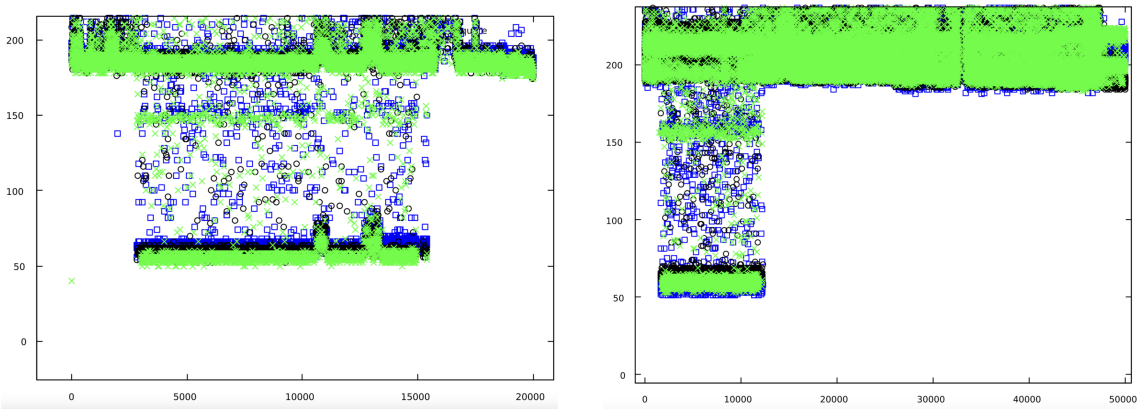
Para complementar os resultados quantitativos apresentados na Tabela 1, os gráficos de tempo de acesso apresentados nas Figuras 2, 3 e 4 possibilitam uma análise visual. A Figura 2 ilustra dois exemplos da execução do ataque em ambiente nativo com `clflush` (Figura 2a no i7-6500U e Figura 2b no i5-7500T). É possível observar uma clara distinção entre duas faixas de tempo de acesso: uma inferior correspondendo aos *cache hits* (indicando acesso da vítima GPG) e uma superior correspondendo aos *cache*

[2]Ticks do contador customizado Eagle Timer, oferece maior resolução que `performance.now()`, embora sua relação exata com nanossegundos possa variar.

**Table 1. Thresholds e resultados de acerto médio da chave RSA.**

| Configuração | Processador | Threshold | Acerto Médio (%) | Variância |
|---|---|---|---|---|
| Nativo com clflush | i7-6500U | 70 ciclos | 49.87 | 0.0160 |
| Nativo com clflush | i5-7500T | 100 ciclos | 49.82 | 0.0001 |
| Nativo com *eviction sets* | i7-6500U | 200 ciclos | 29.69 | 0.0066 |
| Nativo com *eviction sets* | i5-7500T | 170 ciclos | 28.00 | 0.0019 |
| Web com *eviction sets* | i7-6500U | 40 ticks | 3.59 | 0.0001 |
| Web com *eviction sets* | i5-7500T | 60 ticks | 0.12 | 0.0001 |

*misses* (quando a vítima não acessou o endereço monitorado). Essa separação nítida permite uma classificação confiável dos eventos e resulta nas maiores taxas de acerto, próximas a 50%.



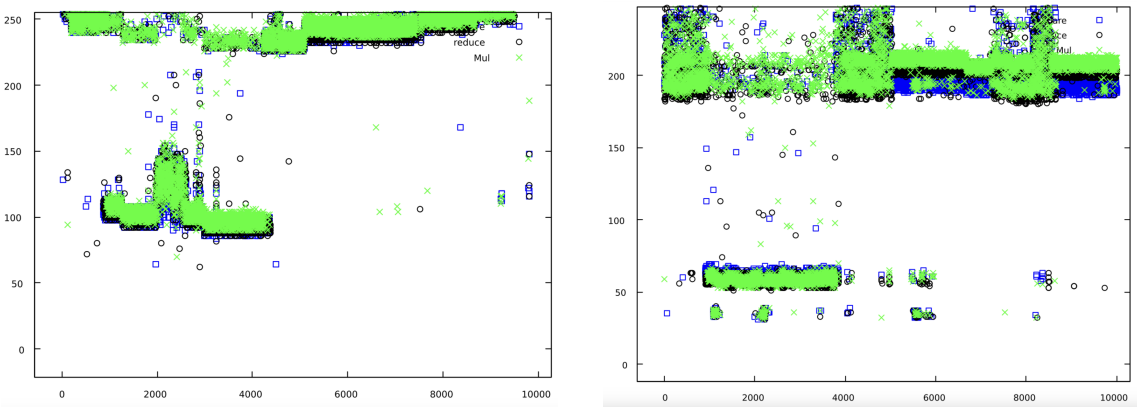

**(a) Nativo com clflush (i7-6500U)** **(b) Nativo com clflush (i5-7500T)**

**Figure 2. Exemplo de tempo de acesso (ciclos de CPU) *vs* amostra para o ataque nativo com `clflush`. Note a clara separação entre tempos baixos (hits) e altos (misses), permitindo a classificação eficaz.**

A Figuras 2a e 2b mostram dois exemplos da execução do ataque no ambiente nativo com *eviction sets*. Embora ainda seja possível distinguir visualmente as faixas de *hits* e *misses*, é possível notar um aumento considerável de ruído em comparação com o uso do `clflush`. Amostras que deveriam ser *misses* podem apresentar tempos de acesso mais baixos (falso *hit*), e vice-versa. Esse ruído adicional dificulta a classificação precisa, refletindo na menor taxa de acerto, em torno de 30%.

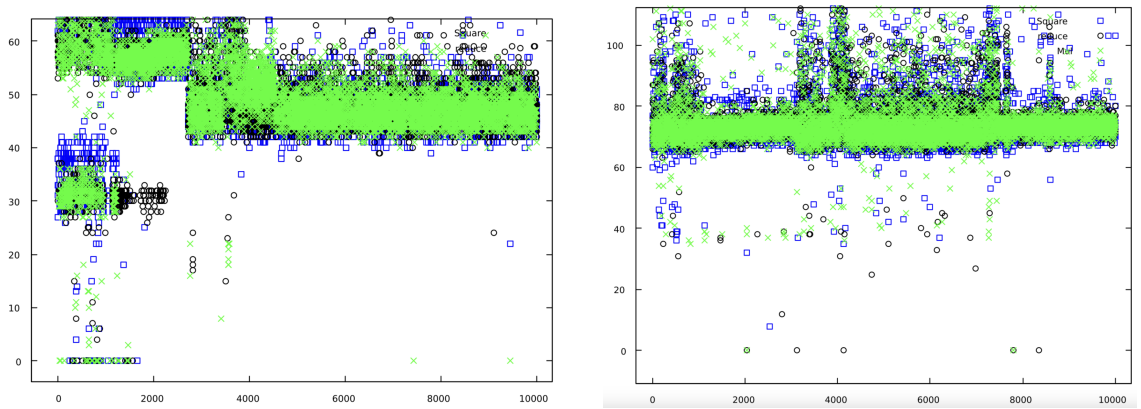
Finalmente, a Figura 4 ilustra um resultado típico do ataque web com *eviction sets*. O gráfico da Figura 4a apresenta um nível de ruído muito elevado e inconsistência entre execuções. Na maioria dos casos, não foi possível identificar visualmente faixas distintas correspondentes a *hits* e *misses* ou mesmo o início e fim da execução do GPG. A tentativa de classificação com os thresholds da Tabela 1 resultou em taxas de acerto residuais (inferiores a 4%), indicando a falha do ataque nesta configuração em extrair informações úteis sobre a chave secreta.

Em resumo, os resultados indicam que ambas as abordagens nativas foram capazes de extrair uma porção significativa dos bits da chave RSA (superando o limiar




**(a) Nativo com *eviction sets* (i7-6500U)** **(b) Nativo com *eviction sets* (i5-7500T)**

**Figure 3. Exemplos de tempo de acesso (ciclos de CPU) *vs* amostra para o ataque nativo com *eviction sets*. Comparado ao uso de `clflush` (Figura 2), é possível observar um aumento considerável de ruído, tornando a separação entre hits (tempos baixos) e misses (tempos altos) menos nítida, embora ainda parcialmente distinguível em algumas execuções, resultando em menor precisão na classificação.**




**(a) Web com *eviction sets* (i7-6500U)** **(b) Web com *eviction sets* (i5-7500T)**

**Figure 4. Exemplos de tempo de acesso (*ticks* do Eagle Timer) *vs* amostra para o ataque web com *eviction sets*. Os gráficos ilustram o alto nível de ruído e inconsistência inerentes à execução no navegador, sem uma separação visual clara entre potenciais hits e misses.**

de 27% considerado relevante pela literatura para recuperação completa da chave com análises adicionais [Yarom and Falkner 2014]), sendo a versão com `clflush` mais precisa e consistente. Em contraste, a adaptação para o ambiente web, utilizando as técnicas implementadas, não obteve sucesso, apresentando taxas de acerto muito baixas e inconsistentes.

## 5. Discussão

Os resultados experimentais apresentados na Seção 4 indicam uma clara disparidade entre a eficácia do ataque Flush+Reload em ambiente nativo e sua adaptação para o ambiente web, nas condições testadas. Enquanto ambas as implementações nativas obtiveram sucesso na recuperação de uma fração significativa dos bits da chave RSA do GPG (aproximadamente 50% com `clflush` e 29% com *eviction sets* , em média), a versão web falhou expressivamente (ficando com acerto abaixo de 4%).

O sucesso da implementação do ataque em ambiente nativo com `clflush` confirma a eficácia da técnica original [Yarom and Falkner 2014], onde o acesso direto à instrução `clflush` e ao temporizador de alta precisão `rdtscp` permitem um monitoramento de baixa granularidade e baixo ruído, refletido na clara separação entre *hits* e *misses* (Figura 2) e na alta taxa de acerto. A implementação do ataque em ambiente nativo com *eviction sets*, embora conceitualmente funcional, apresentou maior ruído e menor acerto (aproximadamente 30%). Isso sugere que, mesmo em ambiente nativo, a simulação do `clflush` via *eviction sets* introduz imprecisões ou lentidão que dificultam o acompanhamento perfeito das operações do GPG, embora ainda produza um sinal suficientemente claro para superar o limiar de recuperação da chave (que é de aproximadamente 25% [Heninger and Shacham 2009]).

A falha da implementação web com *eviction sets* pode ser atribuída a uma combinação de fatores inerentes ao ambiente do navegador e à natureza *cross-environment* do ataque, a saber:

**Limitações de temporização.** Apesar de usar uma técnica avançada como o *eagle timer*, a resolução e a frequência de monitoramento alcançáveis pelo loop WebAssembly/JavaScript podem ser insuficientes para distinguir confiavelmente os tempos de acesso à cache na escala necessária para rastrear as operações do processo GPG nativo, que executa em velocidade máxima fora do navegador. O `rdtscp` nativo opera na ordem de ciclos de *clock*, uma granularidade provavelmente muito mais fina que a obtida via *eagle timer*.

**Ruído do Ambiente.** O ambiente de execução do navegador é intrinsecamente ruidoso. A atividade do próprio motor JavaScript/WebAssembly, renderização, *garbage collection*, outras abas, extensões e a multiplexação de processos pelo sistema operacional introduzem variações significativas nos tempos de acesso à cache, que podem facilmente mascarar as sutis diferenças entre *hits* e *misses* exploradas pelo Flush+Reload. Os gráficos inconsistentes e ruidosos da Figura 4 corroboram com essa hipótese.

**Desafios do *Cross-Environment*:** Monitorar um processo nativo externo a partir do navegador adiciona camadas de complexidade e possíveis fontes de ruído e latência (interação com o sistema operaciona e *scheduling*, por exemplo) que não existem no ataque nativo-nativo.

### 5.1. Implicações do resultados

Ao analisar os resultados no contexto de trabalhos anteriores sobre ataques de cache via web, podem ser realizadas algumas observações. Ataques como os de [Oren et al. 2015], [Gierlichs et al. 2017] e [Shusterman et al. 2021b, Shusterman et al. 2021a] demonstraram ser possível extrair informações via cache a partir do navegador, mas geralmente com resolução temporal menor ou focando em identificar atividades de larga escala (como visitas a sites) ou vulnerabilidades no próprio ambiente do navegador (como quebrar ASLR ou atacar bibliotecas JavaScript). O trabalho de [Genkin et al. 2018], embora utilizando WebAssembly e visando chaves criptográficas, tinha como alvo bibliotecas executando dentro do ambiente do navegador. Nosso trabalho se diferencia por testar a fronteira de um ataque de alta resolução temporal (Flush+Reload) lançado de um ambiente web contra um processo nativo externo.

A falha observada sugere que, com as técnicas atuais e sob as condições testadas, existe um limite prático para a ameaça representada por esta classe específica de ataque de canal lateral via web. Embora o WebAssembly aumente significativamente as capacidades computacionais do navegador, ele não elimina completamente as barreiras de ruído e resolução temporal impostas pelo ambiente para ataques que dependem de medições extremamente precisas e de alta frequência contra alvos externos.

### 5.2. Limitações

Os experimentos foram restritos a duas CPUs Intel específicas e a uma versão particular do GPG. Arquiteturas diferentes (por exemplo, AMD) ou versões mais recentes do GPG/Libgcrypt poderiam apresentar resultados distintos. As técnicas de temporização (*eagle timer*), geração de *eviction sets* e análise de dados (máquina de estados) utilizadas são apenas algumas das abordagens possíveis, sendo que métodos alternativos ou mais sofisticados (como análise manual ou por meio de *machine learning*) poderiam, talvez, extrair mais informações dos sinais ruidosos. Além disso, fatores ambientais não totalmente controlados, como carga do sistema ou configuração específica do navegador, podem ter influenciado nos resultados.

## 6. Conclusão

Este trabalho investigou a viabilidade de executar um ataque de canal lateral de cache com alta resolução temporal, o Flush+Reload, a partir do ambiente web (JavaScript/WebAssembly) contra um processo nativo (GnuPG 1.4.13), comparando-o com implementações nativas. Os resultados experimentais demonstraram que, enquanto as versões nativas do ataque (utilizando `clflush` ou *eviction sets*) conseguiram recuperar uma porção significativa dos bits da chave secreta RSA (com acerto médio entre 28% e 50%), superando limiares relevantes da literatura, a adaptação para o ambiente web falhou em obter resultados consistentes. A taxa de acerto da versão web foi residual (inferior a 4%), insuficiente para a recuperação da chave, e os dados de temporização mostraram-se extremamente ruidosos e inconsistentes.

Concluímos que, apesar das capacidades de desempenho do WebAssembly e do uso de técnicas avançadas de temporização e manipulação de cache adaptadas para o navegador (como *eviction sets* e *eagle timer*), a execução de ataques Flush+Reload de alta resolução a partir do ambiente web contra alvos nativos rápidos enfrenta obstáculos

práticos consideráveis. A frequência de monitoramento alcançável e a precisão temporal parecem insuficientes para capturar os padrões de execução de alvos nativos velozes, além do ruído inerente ao ambiente do navegador mascarar os sinais sutis da cache. Este resultado sugere limites práticos para a exploração cross-environment dessa classe específica de ataques de canal lateral via web, diferenciando-os de outros ataques web de menor resolução ou contra alvos internos ao navegador que foram demonstrados com sucesso na literatura.

Como trabalhos futuros, a aplicação de técnicas de aprendizado de máquina para análise dos sinais ruidosos de tempo ou a exploração de primitivas de temporização ainda mais precisas no ambiente web poderiam ser investigadas na tentativa de superar as limitações encontradas.

## References

(2023). FIPS PUB 186-5: Digital Signature Standard (DSS). FIPS Standard FIPS PUB 186-5, U.S. Department of Commerce, Gaithersburg, MD.

da Silva Simões, F., de Souza Bento, L. M., and Machado, R. C. S. (2021). Uma implementação do ataque flush+reload para recuperação de trechos de chave rsa. In *ANAIS DO WRAC+2020 & WCIBER 2020, 2020, Rio de Janeiro*. Galoá.

Ge, Q., Yarom, Y., Heiser, G., and Armstrong, K. (2015). A survey of microarchitectural timing attacks and countermeasures. *Computers and Security*, 30:3–29.

Genkin, D., Pachmanov, L., Tromer, E., and Yarom, Y. (2018). Drive-by key-extraction cache attacks from portable code. In *Applied Cryptography and Network Security: 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, page 83–102, Berlin, Heidelberg. Springer-Verlag.

Gierlichs, B., Gras, L., Kim, D., and Maurice, C. (2017). Cache side-channel attacks in the JavaScript context. In *Proceedings of the ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, pages 809–822. ACM.

Gordon, D. M. (1998). A survey of fast exponentiation methods. *Journal of Algorithms*, 27(1):129–146.

Heninger, N. and Shacham, H. (2009). Reconstructing rsa private keys from random key bits. In Halevi, S., editor, *Advances in Cryptology - CRYPTO 2009*, pages 1–17, Berlin, Heidelberg. Springer Berlin Heidelberg.

Mazaheri, A., Sarmadi, S., and Ardakani, A. (2022). Analyzing side-channel attacks in webassembly. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE.

Oren, Y., Kemerlis, V. P., Sethumadhavan, S., and Keromytis, A. D. (2015). The spy in the sandbox: Practical cache attacks in JavaScript and their implications. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 1406–1418. ACM.

Shusterman, A., Avraham, Z., Croitoru, E., Haskal, Y., Kang, L., Levi, D., Meltser, Y., Mittal, P., Oren, Y., and Yarom, Y. (2021a). Website fingerprinting through the cache occupancy channel and its real world practicality. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2042–2060.

Shusterman, A., Finkelstein, C., Gruner, O., Shani, Y., and Oren, Y. (2021b). Cache-based characterization: A low-infrastructure, distributed alternative to network-based traffic and application characterization. *Computer Networks*, 200:108550.

Tanenbaum, A. S. and Bos, H. (2014). *Modern Operating Systems*. Prentice Hall Press, USA, 4th edition.

Vila, L., K"opf, M., and Morales, J. (2018). Theory and practice of finding eviction sets. *Proceedings on Privacy Enhancing Technologies*, 2018(3):263–279.

Yarom, Y. and Falkner, K. (2014). Flush+reload: A high resolution, low noise, L3 cache side-channel attack. In *Proceedings of the 23rd USENIX Security Symposium*, pages 719–732. USENIX Association.