SCRDTs: Tipos de Dados Seguros e Replicados sem Conflitos *

Joana Tavares, Nuno Preguiça, and Bernardo Ferreira

FCT, Universidade NOVA de Lisboa & NOVA LINCS
jd.tavares@campus.fct.unl.pt,{nuno.preguica,bf}@fct.unl.pt

Resumo Tipos de dados replicados sem conflitos (CRDTs) têm sido utilizados, tanto na academia como na indústria, em aplicações geo-distribuídas baseadas na núvem para garantir a convergência dos dados. No entanto, recentes incidentes de segurança continuam a levantar questões sobre a confiabilidade de soluções na nuvem e sobre os requisitos necessários para poder utilizar estas soluções com segurança. Neste artigo propomos SCRDTs, um novo tipo de CRDTs que alia segurança e privacidade dos dados às propriedades tradicionais dos CRDTs. Apresentamos o desenho de diferentes tipos de SCRDTs, incluindo SCRDTs para registos, conjuntos, mapas, listas e contadores. Através da adaptação segura de técnicas de Cifra Determinística e Cifras Parcialmente Homomórficas, os SCRDTs conseguem oferecer as funcionalidades dos CRDTs, minimizando a revelação de informação durante a realização de operações. Os resultados obtidos com uma implementação dos desenhos propostos mostra que este apresentam um custo adicional razoável face a uma solução não segura.

Keywords: CRDTs, Sistemas Distribuidos, Processamento de Dados Cifrados

1 Introdução

A área de computação na Nuvem é hoje um tópico importante de investigação, com interessantes aplicações em sistemas distribuídos [1] e bases de dados geo-replicadas [2, 3]. No entanto, recentes incidentes de segurança [4–6] demonstram a fragilidade destas soluções e a generalizada falta de confiança em provedores de serviços na Nuvem. Em particular, destacam-se dois tipos de problemas comuns: disponibilidade e privacidade dos dados.

Os problemas de disponibilidade costumam ser endereçados por técnicas de replicação [2], levantando-se o problema de como manter consistência entre as diferentes réplicas dos dados. Os CRDTs (Tipos de Dados Replicados sem Conflitos) [7] endereçam exatamente este problema, oferecendo aos programadores abstrações simples que garantem a convergência das várias réplicas usando políticas de resolução de conflitos bem-definidas. No entanto a privacidade dos

* Este trabalho foi parcialmente apoioado pela FCT/MCTES, através do projecto #PTDC/CCI-INF/32662/2017, com financiamento do FEDER e do projecto #UID/CEC/04516/2013, e pelo projecto H2020 LightKone (número 732505).

dados, principalmente no contexto da disponibilidade e consistência, continua a ser um grande problema em aberto [8].

Neste artigo propomos SCRDTs (Tipos de Dados Seguros e Replicados sem Conflitos), uma nova família de CRDTs que alia segurança e privacidade dos dados às propriedades de convergência com semântica bem-definida dos CRDTs. O grande desafio resolvido pela nossa proposta consistiu em como suportar (de forma eficiente) todas as operações necessárias em CRDTs, nomeadamente operações de atualização, consulta e fusão de estado, sem comprometer a sua privacidade e segurança. Para tal utilizámos combinações de técnicas convencionais de criptografia simétrica (probabilística e determinística) e técnicas de criptografia parcialmente homomórfica, de forma a suportar as operações necessárias com o mínimo de revelação de padrões e com baixa latência.

Alavancando destas técnicas, desenhámos diferentes CRDTs seguros, incluindo registos, conjuntos, mapas e listas. Estes tipos de CRDTs são essenciais para suportar aplicações web e bases de dados NoSQL. Os SCRDTs desenhados foram implementados e avaliados em termos de segurança e desempenho.

O resto do artigo segue a seguinte estrutura: na Secção 2 discutimos trabalho relacionado mais relevante para este artigo; na Secção 3 apresentamos uma visão de alto nível da solução, assim como os modelos de sistema e adversário considerados; a Secção 4 descreve os detalhes da nossa solução; a Secção 5 discute uma implementação de referência e resultados experimentais preliminares; e a Secção 6 apresenta as conclusões finais do artigo.

2 Trabalho Relacionado

Os CRDTs foram originalmente propostos por Shapiro et al. [7], como forma de garantir a convergência de réplicas modificadas concorrentemente. Estes permitem criar sistemas distribuídos que fornecem elevada disponibilidade, tolerância a falhas e baixa latência. Neste contexto, oferecer adicionalmente condições de segurança refere-se a manter a privacidade e a confidencialidade dos dados.

Os CRDTs possuem dois tipos principais de métodos de replicação, nomeadamente replicação com base no estado ou nas operações. Na nossa solução foi adotada a metodologia de replicação com base no estado, sendo que esta consiste em periodicamente as réplicas de um CRDT propagarem o seu estado entre si e aplicarem uma função de fusão de estado, função *merge*, que obedece a um conjunto de propriedades que garantem que as réplicas convergem para o mesmo estado. Como trabalho futuro, iremos também considerar CRDTs com replicação baseada nas operações.

Face a recentes incidentes de segurança relacionados com aplicações na Nuvem [4–6], a computação sobre dados cifrados tornou-se alvo de investigação

pela comunidade científica, pois não só permitem manter a privacidade dos dados dos utilizadores sem comprometer o correcto funcionamento das aplicações [9] mas também permitem fazer *outsourcing* de computações pesadas para a Nuvem com garantias de confidencialidade.

Existem diferentes tipos de esquemas criptográficos que permitem efetuar computações sobre dados cifrados. Neste trabalho usámos Cifras Determinísticas (CD) e Cifras Parcialmente Homomórficas (CPH) [10], uma vez que permitem implementar a funcionalidade necessária com um custo mínimo de desempenho. As CD permitem efetuar operações de igualdade, podendo expor a frequência dos dados armazenados [11]. As CPH designam esquemas criptográficos onde dada uma chave criptográfica fixa, é possível executar operações de adição ou multiplicação sobre o texto cifrado [12]. Um exemplo clássico é o esquema criptográfico de Paillier [13] utiliza uma técnica de *trapdoor* baseada em teoria de resíduos para permitir a operação de adição sobre os dados cifrados.

Ao longo dos anos foram propostas diferentes soluções que permitem armazenar e efetuar computações sobre dados encriptados como o CryptDB [14] e Depsky [15]. No caso do Depsky, não é possível executar operações sobre os dados cifrados o que limita a sua funcionalidade. Já o CryptDB permite executar operações em bases de dados SQL. Neste trabalho, apresentamos uma solução que permite efetuar operações sobre CRDTs, a qual pode ser usada em bases de dados NoSQL que os utilizem. Para o efeito, adoptamos uma abordagem similar à apresentada por Marcelo et. al. [16] onde são utilizados módulos para suportar a encriptação dos dados e execução de operações sobre os mesmos. Alternativamente, soluções como CipherBase [17] e Cuttlefish [18], combinam esquemas criptográficos com hardware confiável para o mesmo propósito. Nestas, o objetivo do hardware confiável é minimizar a informação revelada e otimizar o desempenho. No futuro, pretendemos estender a nossa solução para permitir a utilização de hardware confiável.

3 Visão Geral do Sistema

Na Figura 1 encontra-se representada uma visão alto nível do sistema onde se pretende integrar os SCRDTs, sendo este um sistema de bases de dados (SBD) do tipo *key-value store* onde os valores associados às chaves serão os SCRDTs. A manipulação (criação, remoção e alterações) destes será feita através de componentes *Stubs*, que se encontram junto dos clientes. Tanto o cliente como os *Stubs* são considerados confiáveis.

Os *Stubs* são responsáveis por gerir chaves criptográficas e cifrar os dados antes de serem enviados para a Nuvem, o que permite aos clientes serem agnósticos às primitivas criptográficas envolvidas. No caso de os dados serem

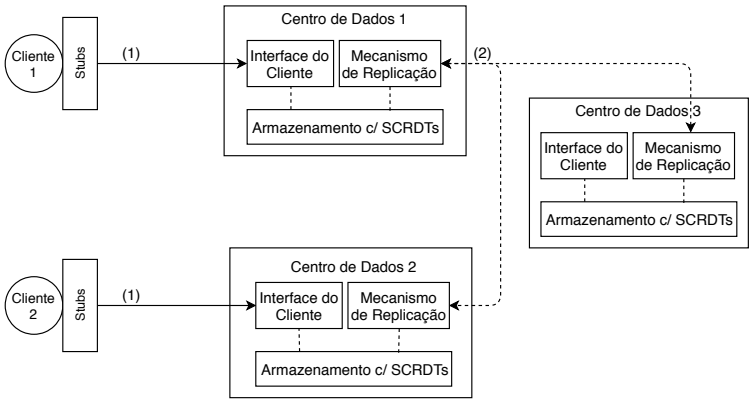

Figura 1: Esquema do modelo de sistema.

partilhados entre clientes, assume-se que estes terão efetuado *à priori* um protocolo de troca de chaves criptográficas e outros parâmetros, assegurando deste modo que todos conseguem manipular os dados. Após estes serem cifrados, a componente *Interface do Cliente* exposta pelo SBD é utilizada pela componente *Stubs* para armazenar os dados na Nuvem.

O SCRDTS e respetivos *Stubs* permitem que a nossa solução possa ser integrada com o SBD de dois modos distintos quanto a onde as operações são executadas: junto do cliente ou do SBD. Ambos requerem obter a estrutura sobre a qual se pretende operar, executar a operação e propagar o resultado para o SBD através das operações *put* e *get*. No caso de serem executadas junto do cliente, a componente responsável por este processo será os *Stubs*. No caso de serem executadas junto do SBD, a componente responsável será a *Inteface do Cliente*, sendo necessário efetuar uma extensão à mesma para este efeito.

Para garantir a convergência dos SCRDTs, a componente *Mecanismo de Replicação* do SBD irá usar a interface exposta pelos mesmos para obter e propagar o estado por todas as réplicas e aplicar a função de *merge* (fusão) aquando da receção de um novo estado.

3.1 Modelo de Adversário

O nosso modelo de adversário considera um administrador da Nuvem do tipo honesto mas curioso que opera a infra-estrutura da Nuvem e os seus servidores. Este adversário tem acesso a todos os dados que passam pela rede e que são armazenados em memória principal ou secundária na Nuvem. O objetivo do adversário é inferir o conteúdo dos dados e das computações efetuadas sobre os mesmos. Também admitimos adversários externos à infraestrutura da Nuvem que pretendem atacar as comunicações efetuadas ao nível interno e externo da infraestrutura da Nuvem, assinaladas com (1) e (2), respetivamente, na Figura 1.

Tabela 1: Estruturas de dados desenvolvidas.

Estrutura	**Política de Resolução de Conflitos**	**Esquema Criptográfico**
Conjunto	*Last Writter Wins*, *Add Wins*	Determinístico (Valor)
Lista	*Last Writter Wins*, *Add Wins*	Determinístico (Valor)
Mapa	*Last Writter Wins*, *Add Wins*	Determinístico (Chave) e Aleatório (Valor)
Registo	*Last Writter Wins*	Aleatório (Valor)
Contador	*Add Wins*	Paillier (Valor)

Tabela 2: Esquemas criptográficos utilizados (* usa-se um vetor de inicialização fixo).

Nome do Esquema	**Algoritmo**	**Tipo de Esquema**
Determinístico	AES/CTR/PKCS5*	Cifra Simétrica Determinística
Aleatório	AES/CTR/PKCS5	Cifra Simétrica Probabilística
Paillier	Algoritmo de Paillier	Cifra Parcialmente Homomórfica

O primeiro dos adversários contemplados será mitigado através do uso dos SCRDTs propostos na nossa solução (Secção 4) enquanto que o segundo é mitigado através do uso de canais de comunicação TLS [10].

4 Desenho da Solução

De entre os vários tipos de CRDTs existentes [19], fizemos uma seleção dos mais importantes para suportar bases de dados NoSQL e aplicações web: listas, mapas, conjuntos, registos e contadores. Na Tabela 1 encontra-se descrito de forma sumária os aspetos chave dos SCRDTs implementados, e na Tabela 2 um resumo dos esquemas criptográficos utilizados. De seguida, para cada uma dos tipos de dados é explicado quais os CRDTs e o porquê dos esquemas utilizados.

Conjuntos Foram implementadas três variantes de CRDTs conjunto: *grow-only* (G-Set), *last-writer-wins* (LWW-Set) e *add wins* (Add Wins Set).

O conjunto G-Set permite apenas adicionar elementos, pelo que não existem operações que possam gerar conflitos ao ocorrem concorrentemente. Não obstante, é uma base útil na construção de outras estruturas mais complexas. Além de adicionar, permite pesquisar por elementos, sendo esta operação feita com base em comparações de igualdade, e efetuar a operação de *merge* que, neste caso, consiste na união de conjuntos. Dado que todas as operações são feitas com base nos valores dos elementos e sendo necessário efetuar comparações de igualdade entre os elementos, a variante segura desta estrutura cifra os elementos com o esquema criptográfico Determinístico (Tabela 2).

O conjunto LWW-Set extende o anterior ao suportar operações de remoção, sendo que um elemento pertence ao conjunto caso a última operação efetuada sobre esse elemento seja uma adição. Neste, cada elemento e tem associado uma *timestamp* ts, para que seja possível aplicar a política *last-writer-wins* em caso de operações concorrentes. Adicionalmente, são mantidos dois conjuntos auxiliares distintos do tipo G-Set, para suportar as operações de adição e remoção de elementos. Em concreto, uma operação de adição insere o par (e, ts) ao conjunto de elementos adicionados, A, e a operação de remoção insere o par (e, ts) ao conjunto de elementos removidos, R. Para pesquisar se um elemento pertence ao conjunto, é verificado se o elemento pertence a A e não pertence R ou, no caso de pertencer a ambos, se o *timestamp* do conjunto A é superior ao do conjunto R. A operação de *merge*, consiste em executar o *merge* dos conjuntos G-Set auxiliares da estrutura. Uma vez que este conjunto é baseado em G-Sets, este apresenta os mesmos requisitos pelo que a versão segura deste CRDT também utiliza o esquema criptográfico Determinístico para cifrar os elementos do conjunto (mas não as *timestamps*).

O conjunto Add Wins Set, também conhecido na literatura por *observed-removed set*, permite fazer as mesmas operações que o conjunto LWW-Set. Todavia, neste as operações são feitas com base num identificador único associado a cada elemento e a política de resolução de conflitos é, como o nome indica, a *add wins* i.e. em situações de operações concorrentes a que prevalece é a operação de adicionar. Assim, segue que os elementos deste conjunto são únicos e é possível que existam elementos repetidos sob a condição de os seus identificadores serem diferentes. Além de um identificador único, cada elemento tem associado o tipo e a *timestamp* da última operação efetuado sobre este. Neste conjunto é também mantida uma estrutura auxiliar que faz um mapeamento entre cada elemento e os identificadores que lhe foram associados.

Aquando uma inserção, são gerados os meta dados associados ao elemento e é inserido na estrutura auxiliar uma entrada para este elemento, caso esta ainda não exista, e o identificador gerado. Na remoção de um elemento, são marcados como removidos todos os identificadores associados a este cuja *timestamp* é estritamente inferior à *timestamp* da remoção. Um elemento pertence ao conjunto se existe um identificador que lhe está associado e que não está marcado como removido. A operação de *merge* consiste na união de ambos os conjuntos. Uma vez que, implicitamente, existe o requisito de ser possível efetuar comparações de igualdade entre os elementos, a variante segura desta estrutura cifra os elementos da mesma com o esquema criptográfico Determinístico.

Lista Esta estrutura tem como subjacente uma versão modificada do CRDT Treedoc proposto por Preguiça et. al. [20]. Originalmente, o TreeDoc consiste

num vetor sequencial partilhado onde a cada elemento do vetor está associado um identificador com as seguintes propriedades:

1. Todos os elementos possuem obrigatoriamente um identificador único.
2. O identificador de um dado elemento, mantém-se constante ao longo da existência da estrutura;
3. Existe uma relação de ordem total entre os identificadores, representada por $<$, consistente com a ordem dos elementos no vetor;
4. Dados dois identificadores quaisquer id_1 e id_2, é sempre possível definir um novo identificador id_3 tal que $id_1 < id_3 < id_2$.

Uma vez que as operações sobre esta estrutura são efetuadas sobre os identificadores, estas propriedades permitem que operações concorrentes comutem entre si [20]. Além de verificar estas propriedades, o identificador é composto sendo formado pela posição em que o elemento terá sido inserido, p, e por um diferenciador, d. Tal é necessário pois nesta estrutura a concorrência contempla dois fatores, a operação e a posição à qual a operação se refere, ao contrário das restantes estruturas. Em particular, é necessário para casos em que são inseridos dois elementos distintos concorrentemente na mesma posição sendo que nestes casos o diferenciador permite manter ambos os elementos e estabelecer uma relação de ordem sobre os mesmos.

Na versão modificada da estrutura, os identificadores estão associados aos elementos e a meta dados. Estes meta dados consistem em informação relativa ao tipo e a *timestamp* da última operação efetuada e permitem que seja possível aplicar a política de resolução de conflitos *add wins*. Nesta estrutura a operação de *merge* consiste na união das listas, sendo que para identificadores em comum é determinado qual o valor que deverá prevalecer em função política de resolução de conflitos. Dado que a operação de pesquisa por um elemento é feita com base no elemento em si, apesar das operações de inserção e remoção serem executadas sobre o identificador, a variante segura desta estrutura mantém apenas os elementos cifrados com o esquema criptográfico Determinístico.

Mapa Para esta estrutura foram implementadas duas variantes da mesma com base em dois tipos diferentes de CRDTs mapa, *add-wins* (AddWinsMap) e *last-writer-wins* (LWWMap), cuja principal diferença consiste na política de resolução de conflitos. Intuitivamente, e como o nome indica, na variante AddWinsMap a política é a *add wins* e na variante LWWMap a política é a *last writer wins*. Para que seja possível a deteção de conflitos, em ambos os mapas existe uma *timestamp* associada a cada objeto guardado, ou seja, os mapeamentos efetuados são do tipo $k \rightarrow (v, ts)$. No caso do mapa LWWMap, é efetuado um mapeamento entre uma chave k e um CRDT registo rg com a política *last writer*

wins para obter o comportamento desejado, simplificando a lógica necessária à implementação da estrutura. No caso do mapa AddWinsMap, para além de uma *timestamp*, é também mantida informação sobre a última operação efetuada sobre o elemento para que seja possível determinar qual é a operação que deverá prevalecer. Nesta estrutura a operação de *merge* consiste na união dos mapas, onde para entradas mapeadas pela mesma chave é determinado qual o valor que deverá prevalecer em função da respetiva política de resolução de conflitos. Dado que todas as operações (inserção, remoção, pesquisa de um elemento, pesquisa de elementos e *merge*) são efetuadas através da chave, o requisito de comparação por igualdade aplica-se apenas a esta. Assim, na variante segura de ambos os mapas, as chaves e valores são cifrados com os esquemas Determinístico e Aleatório respetivamente (Tabela 2).

Registo Para implementar esta estrutura, foi utilizado um CRDT registo com a política de *last-writter-wins* para a resolução de conflitos tal como descrito por Shapiro et. al. [19]. Neste registo, cada atualização está associada uma *timestamp* que é criada no momento em que a operação é recebida por uma réplica. Deste modo é possível estabelecer uma ordenação total entre as atualizações, o que, por conseguinte, permite determinar qual das atualizações é a mais recente e qual o valor a ser mantido. Esta lógica é também a que se aplica quando é efetuada a operação de *merge* entre dois registos. Dado que as operações não são efetuadas com base no valor do registo mas sim com base nas *timestamps*, o valor do registo é cifrados utilizando o esquema criptográfico Aleatório.

Contador A implementação da variante segura da estrutura contador tem como subjacente uma implementação de um CRDT PN-Counter [19] modificado para admitir a soma e subtração de valores arbitrários, sendo que estes valores e o valor do contador encontram-se cifrados com o esquema criptográfico Paillier, e para efetuar a operação de *merge* com base em meta dados mantidos em *plaintext*. Originalmente no CRDT PN-Counter, existem dois vetores distintos, P e N, com tantas entradas quanto o número de réplicas que possuí uma cópia do contador sendo que cada réplica opera apenas sobre as entradas que lhe correspondem. Assim, a operação de incrementar um valor arbitrário p numa réplica com o identificador i, consiste em adicionar o valor p à entrada i do vetor P e a operação de incrementar um valor arbitrário n numa réplica com o identificador i, consiste em adicionar o valor n à entrada i do vetor N. O valor final do contador é dado pela diferença entre os somatórios das entradas dos vectores P e N e operação de *merge* consiste no cálculo do máximo de cada entrada para os mesmos vetores. Na versão modificada deste CRDT, as operações de incrementar e decrementar para além de modificarem a respectiva entrada da réplica nos vetores P e N, também alteram a respetiva entrada da réplica em

estruturas auxiliares associadas a cada vetor, MP e MN respetivamente, por um valor estritamente positivo gerado aleatoriamente. Por conseguinte, a operação de *merge* consiste em manter o valor de cada entrada dos vetores P e N associados ao valor máximo de cada entrada nas estruturas auxiliares MP e MN respetivamente. Este artificio é necessário uma vez que os valores do contador estão cifrados segundo o esquema criptográfico de Paillier e este não permite efetuar operações de comparação sobre os mesmos pelo que a operação de *merge* original não seria possível.

4.1 Análise de Segurança

De um modo geral as estruturas que utilizam o esquema Determinista, e que não admitam repetições, gozam de maior segurança quando se encontram *at rest* (i.e., quando não são executadas operações sobre mesmas). Nestes casos, quantas mais operações são executadas mais informação sobre os dados é revelada, como por exemplo informação sobre a pertença ou repetição de um elemento numa estrutura. Este fator é uma consequência direta do determinismo. A repetição de elementos levanta maior vulnerabilidade na estrutura lista em particular uma vez que esta, ao contrário dos mapas e listas, admite este cenário. Em contraste, o uso de um esquema determinista nas estruturas lista e mapa confere o mesmo nível de segurança que o esquema aleatório. As estruturas que utilizam os esquemas Paillier e Aleatório não apresentam estas vulnerabilidades. Uma solução para este problema passa por refrescar ciclicamente todo o texto cifrado ao re-encriptar o mesmo com novos parâmetros. Todavia, dependendo da quantidade de dados presentes nas estruturas e a quantidade de estruturas em si, esta solução pode não ser totalmente prática.

Não obstante existe informação que será sempre observável, nomeadamente informação que provém de meta dados das estruturas que são mantidos em claro como por exemplo os instantes em que operações ocorreram e qual a última operação efetuada. Este tipo de informações dificilmente se consegue esconder uma vez que um atacante que tenha acesso à máquina física onde são efetuadas as computações sobre os dados, atacante este que admitimos no nosso modelo de adversário, consegue observar e deduzir esta informação.

5 Avaliação Experimental e Análise

A avaliação da nossa solução quanto ao desempenho dos SCRDTs, consistiu em efetuar micro-testes dos mesmos e dos respetivos CRDTs em ambiente local. Futuramente, pretende-se efetuar estes testes remotamente para que sejam mais próximos de uma situação real. Os micro-testes consistiram em, para todas as estruturas, executar mil vezes as operações por estas disponibilizadas

Tabela 3: Micro-benchmark com mapas (valores em μs).

Operação	CRDT LWW	SCRDT LWW	CRDT AddWins	SCRDT AddWins
Put	4.77	319.71	3.84	287.8
Remove	3.33	226.6	3.56	289.42
Get	2.79	216.59	2.93	340.2
Contains	2.63	193.54	3.26	209.51
GetAll	156.77	20967.17	175.37	20493.51
Merge	147.91	243.56	178.75	282.94

e medir a média do tempo que cada uma demora a executar. Nas vertentes SCRDTs, o tempo de execução inclui o tempo necessário para (des)serializar e (des)encriptar os dados. Os parâmetros para cada uma das operações, inteiros para os contadores e strings para as restantes estruturas, foram gerados aleatoriamente sendo que no caso das operações de consulta e remoção, as estruturas são devidamente inicializadas com dados antes destas executarem. Cada micro teste foi executado 3 vezes de forma independente e, como tal, os resultados sumariados nas tabelas 3 a 7 refletem uma média das medições retiradas em micro segundos arredondas às centésimas.

5.1 Análise

Globalmente, e como seria expectável, o desempenho dos SCRDTs é pior que o respectivo CRDT salvo exceções. Esta expectativa baseia-se no facto de que nos SCRDTs é tido em conta os tempos de (des)serializar e (des)encriptar os dados.

Uma outra observação geral é o facto de que a operação de consulta de todos elementos em todas as estruturas apresenta valores de tempo de execução elevados. Tal justifica-se por ser necessário iterar as estruturas que representam o estado para calcular os elementos que efetivamente se encontram presentes na estrutura. Nas vertentes seguras, existe ainda a penalização de converter a estrutura em uma outra que contém os elementos no seu tipo de dados original.

No caso da estrutura lista, verifica-se que a operação de remoção apresenta valores muito semelhantes entre as vertentes seguras e não seguras enquanto as restantes operações apresentam valores mais dispares. Tal acontece pois esta operação não requerer encriptar ou desencriptar pois é feita exclusivamente com base no índice do elemento a ser removido. É ainda de notar algumas anomalias como o facto de que na maioria das estruturas a operação de *merge* é significativamente pior na vertente segura quando seria expectável que os valores se mantivessem semelhantes como acontece no caso da estrutura registo.

Tabela 4: Micro-benchmark com conjuntos (N/A = Não aplicável, valores em μs)).

Operação	**CRDT GSet**	**SCRDT GSet**	**CRDT LWW**	**SCRDT LWW**	**CRDT AddWins**	**SCRDT AddWins**
Add	88.09	270.733	105.28	277.12	197.83	283.04
Remove	N/A	N/A	2.18	275.19	10.6	303.99
Contains	1.18	281.33	1.82	257.93	4.29	300.65
getAll	1.26	15334.03	1.32	15417.45	173.81	18338.3
Merge	44.68	131.85	94.3	106.45	272.29	380.24

Tabela 5: Micro-benchmark com listas (valores em μs).

Operação	**CRDT LWW**	**SCRDT LWW**	**CRDT AddWins**	**SCRDT AddWins**
Insert	161.35	297.9	177.95	290.95
Remove	184.61	186.31	180.21	181.4
Get	39.24	109.52	45.29	127.98
Contains	20.89	253.9	20.80	264.84
GetAll	61.97	15233.8	60.97	14958.62
Merge	18305.38	18660.82	17981.87	18671.51

6 Conclusões

Neste artigo propomos um novo tipo de CRDT, os SCRDTs, que estendem as propriedades dos CRDTs com garantias de segurança e privacidade dos dados. Apresentámos adicionalmente o desenho dum leque variado de SCRDTs, que combinam técnicas convencionais de criptografia simétrica e de criptografia parcialmente homomórfica, de forma a revelar o mínimo de informação com um custo adicional razoável, como comprovado na avaliação experimental efetuada. Como trabalho futuro, planeamos construir uma base de dados NoSQL segura e geo-replicada suportada pelos SCRDTs desenhados e desenhar uma outra vertente dos SCRDTs cuja replicação é feita com base nas operações. Adicionalmente, iremos estudar a utilização de hardware confiável moderno baseado em atestação remota, de forma a minimizar ainda mais a revelação de informações durante a execução de operações.

Referências

1. Verissimo, P., Bessani, A., Pasin, M.: The TClouds Architecture: Open and Resilient Cloud-of-clouds Computing. In: DSN-W'12. (2012)
2. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. In: SOSP. (2007)

Tabela 6: Micro-benchmark com registos LWW (valores em μs).

Operação	CRDT	SCRDT
Put	6.37	179.61
Get	2.33	46.47
Merge	0.6	0.6

Tabela 7: Micro-benchmark com contadores (valores em μs).

Operação	CRDT	SCRDT
Inc	2.73	2237.68
Dec	1.57	2319.05
Get	16.57	3274.18
Merge	11.01	12.84

3. Li, C., Porto, D., Clement, A., Gehrke, J., Preguiça, N.M., Rodrigues, R.: Making geo-replicated systems fast as possible, consistent when necessary. In: OSDI. Volume 12. (2012) 265–278
4. Greenwald, G., MacAskill, E.: NSA Prism program taps in to user data of Apple, Google and others. The Guardian (2013)
5. Lewis, D.: iCloud Data Breach: Hacking And Celebrity Photos. Forbes (2014)
6. Cook, T.: A Message to Our Customers. Apple (2016)
7. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: Symposium on Self-Stabilizing Systems, Springer (2011) 386–400
8. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB : Processing Queries on an Encrypted Database. Communications of the ACM (CACM) **55**(9) (2012) 103–111
9. Ohrimenko, O., Schuster, F., Fournet, C., Mehta, A., Nowozin, S., Vaswani, K., Costa, M.: Oblivious multi-party machine learning on trusted processors. In: USENIX Security Symposium. (2016) 619–636
10. Katz, J., Lindell, Y.: Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series). Chapman & Hall/CRC (2007)
11. Ferreira, B.: Privacy-preserving efficient searchable encryption. (2016)
12. Fontaine, C., Galand, F.: A survey of homomorphic encryption for nonspecialists. EURASIP Journal on Information Security **2007**(1) (2007) 013801
13. Paillier, P., Pointcheval, D.: Efficient public-key cryptosystems provably secure against active adversaries. In: Asiacrypt. Volume 99., Springer (1999) 165–179
14. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: Cryptdb: Processing queries on an encrypted database. Commun. ACM **55**(9) (September 2012) 103–111
15. Bessani, A., Correia, M., Quaresma, B., André, F., Sousa, P.: Depsky: dependable and secure storage in a cloud-of-clouds. ACM Transactions on Storage (TOS) **9**(4) (2013) 12
16. Macedo, R., Paulo, J., Pontes, R., Portela, B., Oliveira, T., Matos, M., Oliveira, R.: A practical framework for privacy-preserving nosql databases. In: SRDS, IEEE (2017) 11–20
17. Arasu, A., Blanas, S., Eguro, K., Kaushik, R., Kossmann, D., Ramamurthy, R., Venkatesan, R.: Orthogonal security with cipherbase. In: CIDR. (2013)
18. Savvides, S., Stephen, J.J., Ardekani, M.S., Sundaram, V., Eugster, P.: Secure data types: a simple abstraction for confidentiality-preserving data analytics. In: Proceedings of the 2017 Symposium on Cloud Computing, ACM (2017) 479–492
19. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: A comprehensive study of convergent and commutative replicated data types. PhD thesis, Inria–Centre Paris-Rocquencourt; INRIA (2011)
20. Preguica, N., Marques, J.M., Shapiro, M., Letia, M.: A commutative replicated data type for cooperative editing. In: ICDCS, IEEE (2009) 395–403