# Security and Privacy Challenges: Cloud

Diogo Afonso Costa, Daniel Maia, and Vitor Castro

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
e-mail: {a78034,a77531,a77870}@alunos.uminho.pt

**Abstract.** Desde a sua conceção nos anos 70, a Computação em Cloud tem vindo a tornar-se numa indústria de grande escala usufruída tanto pelas grandes empresas como para uso individual. Naturalmente, com este crescimento surgem várias preocupações, principalmente em relação à proteção dos dados bem como as informações pessoais dos seus utilizadores.

## 1 Introdução

## 2 Principais Desafios Associados

## 3 Cloud computing and cloud storage

### 3.1 Definição de Cloud

Cloud computing nasce da combinação da computação tradicional com a rede. A base deste modelo é a intenção de reduzir o processamento nos terminais tradicionais, passando este a ser feito na Cloud. Deste modo, os utilizadores têm à sua disposição grande poder computacional sem a necessidade de terem que comprar e manter um sistema poderoso. Naturalmente, da Cloud Computing deriva a Cloud Storage, que permitirá o acesso e utilização dos dados, aplicando a mesma lógica base da Cloud Computing. Assim, um grande número de dispositivos de armazenamento poderão estar conectados entre si, funcionando como um cluster, em que os dados estarão distribuídos.

### 3.2 Principais vantagens de uma Cloud

On-demand self-service: cada utilizador tem à sua disposição a quantidade exata de poder computacional (uso de servidores ou de armazenamento) que necessita, sem ter que dar algo de volta ao cloud provider. Broad network access: os utilizadores não têm que se preocupar com as plataformas e dispositivos com que estão a transmitir os dados, uma vez que o £canal de comunicação£ será a Internet com um mecanismo standard definido pela cloud. Esta capacidade permite o uso de vários dispositivos IoT que, não partilhando os mesmos protocolos e interfaces, conseguem comunicar. Resource pooling: capacidade de partilhar recursos e atribuir outros a cada um dos vários clientes, dependendo da sua necessidade atual, dinamicamente, como memória, largura de banda, etc. Rapid elasticity: facilidade de escalar a rede de processamento e armazenamento, o que se traduz numa virtual infinitude de capacidade, para o utilizador. Measured servisse: controlo de uso do serviço, o que permite um pagamento adequado às necessidades do utilizador mas também a recolha de dados de uso para o prestador de serviço.

### 3.3 Principais desafios à construção de uma Cloud confiável

Outsourcing: Perda de controlo da posse dos dados. A Cloud fornecedora de serviço deve ser confiável e providenciar toda a segurança possível, no processamento e armazenamento de dados. A confidencialidade, integridade e não violação dos dados deve ser garantida.

Multi-tenancy: Como plataforma partilhada, a plataforma será partilhada com vários utilizadores, o que pode levar à colocação de dados no mesmo local físico que outros clientes, incorrendo no risco de um programa malicioso tentar manipular esses dados. É necessário, por isso, criar estratégias de proteção de partilha de dados. Massive data and intense computation: Os mecanismos de segurança normais não são suficientes quando tamanha quantidade de dados e computação está disponível. É preciso, por isso, criar novos protocolos e mecanismos de proteção.

### 3.4 Ecossistema de segurança e privacidade de uma Cloud

Confidencialidade: Dados estão disponíveis para o Cloud provider e só um sentido de responsabilidade protege o consumidor. Integridade Disponibilidade Responsabilidade Preservação de privacidade £ privacidade está diferenciado da segurança porque este é um grande tema ao redor da Cloud e um dos seus maiores problemas

### 3.5 Vulnerabilidades de uma Cloud

Co-residence: Partilha de uma mesma estrutura física por diferentes aplicações. Pode haver Cross-VM attack e Malicious SysAdmin. Loss of Physical Control: Os dados e serviços do cliente já não mais estão ao seu alcance, não podendo ser diretamente evitadas manipulação de dados ou software. Bandwidth Under-provisioning: Ataques DOS também se realizam em Clouds, sendo que a capacidade real da rede é bastante inferior à capacidade de todos os serviços da mesma subrede agregados. Cloud Pricing Model: Ataques que promovem a modificação do modelo de imposição de preços, de modo a que um utilizador/serviço pague realmente menos do que era suposto.

### 3.6 Cloud Integrity (foco de abordagem)

Integridade na Cloud implica que não deve acontecer manipulação de dados ou programas, quer por malware ou utilizadores nocivos e, caso esta aconteça, deve ser detetada.

**Ameaças** Data loss / Manipulation: No armazenamento em Cloud são armazenados dados e, como é natural aos servidores existe um risco de perda ou modificação de dados, quer intencionalmente, quer acidentalmente. Erros de administração (restores, data migration, etc) ou ataques propositados para causar a perda de controlo de dados. Dishonest computation in remote servers: Podem acontecer de na computação em Cloud, como os utilizadores não vêem o processamento, serem devolvidos valores/dados errados para um programa ou pedido.

**Defesas** Provable Data Possession: Num ambiente de Cloud não é possível fazer o mesmo tipo de verificação que num tradicional, pois o custo de operações tradicionais seria muito grande em termos de computação (pois os dados armazenados são também eles muito grandes) e largura de banda. Deste modo, torna-se obrigatório falar de Naive Method, Original Provable Data Possession e Proof of Retrievability.

## 4 Propostas Relevantes na Área

### 4.1 O método ingénuo

Um dos possíveis métodos que permite confirmar a integridade dos dados que são carregados/descarregados da cloud, isto é, se estes permanecem os mesmos ou foram alterados, passa por computar o valor da hash do ficheiro correspondente e confirmar se este é o mesmo na origem e no destino.

Efetivamente, o processo pode ser descrito por passos mais detalhados [1] [2]. Se possuirmos um ficheiro F e uma chave aleatória k, podemos usar uma função de hash h para gerar um valor r que identifica unicamente aquele ficheiro no estado em que se encontra. De seguida o ficheiro pode ser "outsourced" (i.e., para a cloud). Sendo estes pré-requisitos satisfeitos o cliente pode a qualquer momento enviar a chave k, que foi usada para calcular o valor r, para o serviço onde o ficheiro foi armazenado e requerir uma nova computação da função h com k e F como argumentos. Se tudo correr de acordo com o previsto, o valor r' que resultará deste cálculo deverá ser igual a r, de onde concluímos que não houve alterações no ficheiro.

Este método pode ser aprimorado [1] [2], na prespectiva em que é possível para um mesmo ficheiro f ter vários valores de r correspondentes à chamada da função h com diferentes valores de k, ou seja, podemos efetuar múltiplos "checksums", totalmente independentes, guardando apenas diferentes chaves assim como valores de hash. Uma outra vantagem surge da forte prova que este método nos proporciona de que o ficheiro F se encontra realmente na cloud/servidor.

Contudo, existe várias desvantagens que tornam este método muitas vezes impraticável [1] [2].

Primeiramente, é necessário que sejam armazenados do lado do cliente todos os valores hash (r) assim como as respetivas chaves (k), sendo o seu número proporcionalmente linear ao número de "checksums" que se pretende fazer. Imaginando que um determinado cliente está a recorrer a um serviço cloud é provável que o seu equipamento possua pouca capacidade de computação ou mesmo de memória, levando a que o armazenamento destas duas componentes possa ser um problema [1] [2].

Além disso, uma outra desvantagem significativa prende-se no facto de ser preciso fazer o processamento de todo o ficheiro F sempre que queremos confirmar se determinado número de hash corresponde ao que o cliente tem na sua posse. Para ficheiros com um tamanho considerável começa a ser notável o peso da computação necessária em todo o processo de confirmação da integridade dos dados quando estes se encontram "outsourced" [1] [2].

Mais ainda, quando procedemos a múltiplas verificações, o ficheiro é inteiramente computado todas as vezes que chamarmos a função de hash com uma nova chave. Obtemos assim um "overhead" totalmente indesejável [1] [2].

Existem algoritmos responsáveis pela verificação da integridade dos dados assim como por autenticar mensagens chamados "message authentication code" (MAC) [3] , em que uma das suas variantes utiliza determinadas funções de hash criptográficas para o efeito, nomeadamente os "Hash-based Message Authentication Code" (HMAC).

As funções de hash criptográficas são funções que devem respeitar cinco características [4] [5]:

1. Compressão - uma função de hash *h* deve converter um input *x* de tamanho finito e arbitrário para um output de tamanho fixo.
2. Facilidade na computação - *h(x)* deve ser fácil de computar.
3. Resistência à pré-imagem - para todos os outputs deve ser inviável que se consiga encontrar o input que originou determinado output.
4. Resistência à segunda pré-imagem - deve ser computacionalmente inviável que dado um input *x'* se consiga encontrar um input *x*, tal que *h(x) = h(x')*.
5. Resistência à colisão - é computacionalmente inviável encontrar dois inputs *x* e *x'* que tenham o mesmo output, i.e. *h(x) = h(x')*.

Estes métodos caracterizam-se por terem sempre associadas chaves secretas de forma a identificar o legítimo emissor dos dados.

O algoritmo HMAC [6] pode ser usado com qualquer função de hash criptográdica, por exemplo MD5 (HMAC-MD5) e SHA-1 (HMAC-SHA1). É de realçar que a segurança e integridade do HMAC depende de vários componentes que devem ser tidos em conta, nomeadamente, como a boa implementação do algoritmo em si, uma escolha aleatória

da chave secreta, um mecanismo seguro de troca de chaves, uma atualização frequente das mesmas, assim como uma sua proteção cuidada. Além disso, o poder criptográfico do HMAC depende das propriedades da função de hash usada, sendo que uma função de hash que tenha sido demonstrada como vulnerável deve ser evitada, por forma a não comprometer a segurança do algoritmo.

### 4.2 PDP - Provable Data Possession

*Provable Data Possession* (PDP) é um processo de verificação da integridade do armazenamento na cloud que assegura os utilizadores que os dados guardados nela estão seguros. Este é diferente de métodos tradicionais no modo em que permite verificação acedendo a pequenas porções de ficheiros em vez os percorrer na sua totalidade.[**?**] O modelo base de PDP requere que os dados sejam pré-processados para que meta-dados sejam guardados pelo cliente para razões de verificação. Se o cliente necessitar de verificar a integridade dos dados, envia um *challenge* ao servidor da *cloud*, solicitando informação sobre um bloco aleatório de dados. O servidor, por sua vez, responde com uma mensagem baseada nos seus conteúdos. Comparando os meta-dados locais com a mensagem, é possível concluir o estado de integridade dos dados. Este modelo permite uma forma eficiente de guardar a integridade dos dados, no entanto, apenas suporta ficheiros estáticos, e devido à sua natureza probabilística, são possíveis falsos positivos.[**?**] A partir desta base, é possível desenvolver formas mais especializadas de PDP, tais como o PDP Escalável, cuja encriptação que reduz o overhead de computação e permite (um número limitado) operações dinâmicas, à custa de restringir o número de clientes com capacidade de acesso. Ainda mais, todos os *challenges* e respostas são pré-computados, limitando o número de possíveis casos, reduzindo a segurança por eles dada[**?**]. Existe também o PDP Dinâmico, que suporta operações dinâmicas na totalidade (inserção, modificação e remoção de ficheiros), mas como resultado causa maior overhead de computação, armazenamento e comunicação com o sevidor. Apesar deste maior overhead, o PDP Dinâmico ainda é relativamente eficiente. Como exemplo, a geração de meta-dados de um ficheiro de 1 GB produz apenas 415 KB e 30 ms de overhead computacional[**?**].

### 4.3 POR - Proof of Retrievability

Atua de modo similar ao PDP. No caso, o POR tenta diminuir a quantidade de processamento e dados envolvidos, pela geração de uma key que codificará um determinado ficheiro F. De seguida, este ficheiro já encodificado será enviado para guardar na Cloud. São também adicionadas sentinelas, que são blocos indistinguíveis que, mais tarde, serão usadas para o controlo. O protocolo POR é ainda capaz de recuperar ficheiros pouco corrompidos. Só aplicável a ficheiros estáveis.
– fim do uso do T10-062..
– inicio do uso do literature_survey£
O POR não obriga à manutenção da totalidade do ficheiro mas requere que, quando necessário, seja devolvida a informação que é pedida. Pode, para além da codificação do ficheiro original, ser mantida parte do ficheiro na máquina local. A grande vantagem do POR em relação ao PDP é que não necessita da devolução do ficheiro por inteiro, o que no caso de ficheiros muitos grandes é significativo. Como não é a totalidade do ficheiro verificada, esta verificação é probabilística. Quando se deseja efetuar a verificação, o utilizador pede à Cloud um conjunto de posições, que sabe serem sentinelas, e compara os valores devolvidos com aqueles que conhece. Caso tenham havido alterações, a sentinela terá sido também manipulada. Como pode acontecer de haverem mudanças que não tenham afetado as sentinelas, o protocolo também possui códigos de correção de erros, corrigindo ficheiros corrompidos. Ao POR, para além da redução do esforço computacional da Cloud, também se garante encriptação de dados. O downside desta encriptação é que é exigido ao computador do utilizador um esforço computacional que não é exigido no PDP.

## 5 Âmbitos de Aplicação

Os exemplos aplicacionais dentro da área da integridade dos dados relacionados com o método PDP e todos os seus derivados são escassos e muitas vezes talvez até mantidos em sigilo.

No entanto, no campo do método *naive* os exemplos são bastantes, ao contrário do que seria de esperar. De facto, o algoritmo HMAC é bastante usado em pedidos onde uma assinatura seja necessária, por exemplo quando é precisa uma autenticação. É importante perceber que neste contexto, assinatura significa algo que demonstra que quem está a realizar determinada ação é legítimo de a fazer.

Assim sendo, e tomando como exemplo a Amazon Web Service (AWS), percebemos que os serviços disponibilizados têm à sua disposição assinaturas HMAC-SHA256. O processo é bastante simples. Primeiramente, o cliente elabora um *resquest*. De seguida é criada a assinatura com base na HMAC-SHA e numa chave secreta que só o cliente e o respetivo servidor é que possuem. Depois é enviado para os servidores o *request* juntamente com a devida assinatura. No servidor, é procurada a chave secreta e consequentemente computada a assinatura com base no *request* recebido. Por fim as assinaturas são comparadas e o *request* só é validado se estas foram iguais (Figura 1).

De facto, assim como a AWS também outras empresas multinacionais como é o caso da Google [7] ou da Alibaba Cloud [8] utilizam/disponibilizam este mecanismo de assinatura.

Além disso, é de constatar que um grande número de empresas internacionais disponibilizam simples funções de hash criptográficas com o objetivo de verificar a integridade dos dados que são enviados ou descarregados das suas respetivas clouds.

Nomeadamente, o Git é um dos sistemas que mais intrinsecamente tem implementado um sistema de verificação de integridade de dados [9]. Efetivamente, todo e qualquer ficheiro que é guardado neste sistema é feito um checksum e a partir desse momento o ficheiro é tratado pelo seu código de hash e não pelo seu nome. Estando esta funcionalidade implementada até nos níveis mais baixo, possibilita a que nenhum documento seja alterado ou corrompido sem que o Git saiba. Para o efeito é usada a função de hash SHA-1. Esta função produz como resultado uma *string* com quarenta caracteres, por isso é que constantemente nos cruzamos com *strings* do tipo 24b9da6552252987aa493b52f8696cd6d3b00373 quando usamos este sistema de gestão de versões.

Assim como o Git muitas outras empresas utilizam o mesmo sistem de uma forma mais ou menos análoga, por exemplo, a Google Cloud Strorage [10], a Amazon Simple Storage Service (Amazon S3) [11], onde ambas utilizam a função de hash MD5 como forma de verificar a integridade dos dados quandos estes são carregados ou descarregados dos serviços cloud.

## 6 Conclusão

Neste trabalho...

## References


1. Xiao, Z., Xiao, Y.: Security and privacy in cloud computing. IEEE Communications Surveys Tutorials **15**(2) (2013) 843–859
2. Juels, A., Kaliski, Jr., B.S.: Pors: Proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. CCS '07, New York, NY, USA, ACM (2007) 584–597
3. Wikipedia: Autenticador de mensagem. `https://pt.wikipedia.org/wiki/Autenticador_de_mensagem` (2017) [Online; acedido a 30-Setembro-2017].
4. Mathews, M.M., V, P.: Date time keyed - hmac. In: 2016 Online International Conference on Green Engineering and Technologies (IC-GET). (2016) 1–5

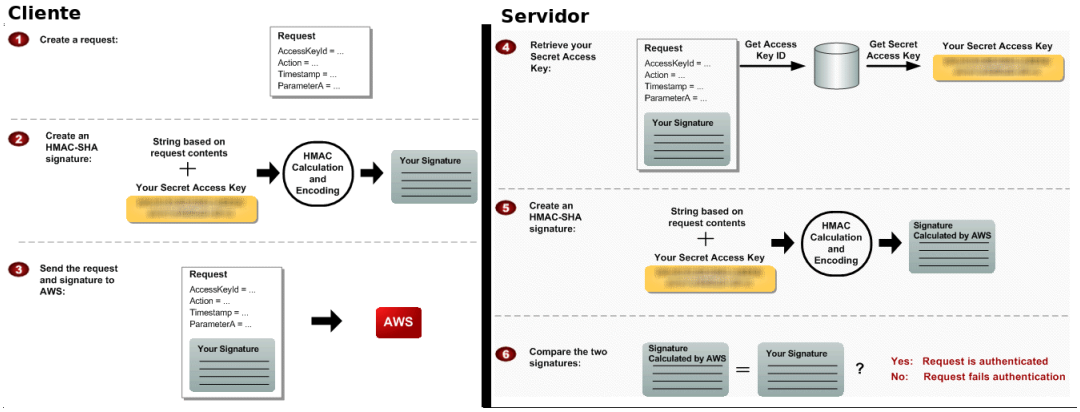


**Fig. 1.** Exemplo de utilização do HMAC.


5. Wikipedia: Hmac. https://pt.wikipedia.org/wiki/HMAC (2017) [Online; acedido a 30-Setembro-2017].
6. Krawczyk, H., Bellare, M., Canetti, R.: Hmac: Keyed-hashing for message authentication (1997)
7. Inc., G.: Migrating from amazon s3 to google cloud storage. https://cloud.google.com/storage/docs/migrating (2017) [Online; acedido a 30-Setembro-2017].
8. Inc., A.: Signature mechanism. https://www.alibabacloud.com/help/doc-detail/26051.htm (2017) [Online; acedido a 30-Setembro-2017].
9. Git: 1.3 getting started - git basics. (https://git-scm.com/book/en/v2/Getting-Started-Git-Basics) [Online; acedido a 30-Setembro-2017].
10. Inc., G.: Using google cloud storage with big data. https://cloud.google.com/storage/docs/working-with-big-data (2016) [Online; acedido a 30-Setembro-2017].
11. Inc., A.W.S.: How do i ensure data integrity of objects uploaded to or downloaded from amazon s3? https://aws.amazon.com/premiumsupport/knowledge-center/data-integrity-s3/ (2016) [Online; acedido a 30-Setembro-2017].