Монолитный криптографический протокол

Коваленко Геннадий Александрович

Аннотация. Монолитность протокола, определяемая в первую очередь его самодостаточностью, сводится также к его имманентности. Последнее свойство является уникальным качеством класса подобных протоколов, потому как содержит всю информацию внутри шифрованной оболочки. Чтобы иметь представление маршрута передаваемой информации, каждый субъект самолично пытается её расшифровать. Безуспешность расшифрования лишь свидетельствует о факте непричастности данного объекта к текущему субъекту получателю. Таким образом на уровне протокола автоматически производится постоянная авторизация субъектов к хранимой, либо передаваемой информации.

Ключевые слова: монолитный криптографический протокол; протокол Bitmessage; скрытые системы; программная реализация;

Содержание

1. Введение

Ядром всех скрытых систем [1] являются криптографические протоколы. Наиболее приоритетными протоколами, в конечном счёте, становятся простые, легкочитаемые и легкореализуемые. В массе своей, практические составляющие реального мира часто приводят к необходимости выбирать компромиссы между теоретической безопасностью и практической производительностью. Тем не менее, существуют протоколы стремящиеся к теоретической безопасности, но при этом не исключающие практическую производительность для малых групп участников. К такому виду протоколов может относится монолитный криптографический протокол, являющийся одновременно наследником протокола Bitmessage [2] и классом его выражения. Главной особенностью протокола становится его самодостаточность [3, с.80] и простота [3, с.58], а также абстрактность, за счёт которой появляется возможность применять данный протокол в тайных каналах связи и во множестве анонимных сетей.

2. Определение

Протокол определяется восьмью шагами, где три шага на стороне отправителя и пять шагов на стороне получателя. Для работы протокола необходимы алгоритмы КСГПСЧ

(криптографически стойкого генератора псевдослучайных чисел), ЭЦП (электронной цифровой подписи), криптографической хеш-функции, установки / подтверждения работы, симметричного и асимметричного шифров.

Участники протокола:

A - отправитель,

B - получатель.

Шаги участника A:

1. $K = G(N), R = G(N)$,

   где G - функция-генератор случайных байт,
   N - количество байт для генерации,
   K - сеансовый ключ шифрования,
   R - случайный набор байт.

2. $H_P = H(R || P || PubK_A || PubK_B)$,

   где H_P - хеш сообщения,
   H - функция хеширования,
   P - исходное сообщение,
   $PubK_X$ - публичный ключ.

3. $C_P = [E(PubK_B, K), E(K, PubK_A), E(K, R), E(K, P), H_P, E(K, S(PrivK_A, H_P)), W(C, H_P)]$,

   где C_P - зашифрованное сообщение,
   E - функция шифрования,
   S - функция подписания,
   W - функция подтверждения работы,
   C - сложность работы,
   $PrivK_X$ - приватный ключ.

Шаги участника B:

4. $W(C, H_P) = P_W(C, W(C, H_P))$,

   где P_W - функция проверки работы.
   Если ≠, то протокол прерывается.

5. $K = D(PrivK_B, E(PubK_B, K))$,

   где D - функция расшифрования.
   Если ≠, то протокол прерывается.

6. $PubK_A = D(K, E(K, PubK_A))$.

   Если ≠, то протокол прерывается.

7. $H_P = V(PubK_A, D(K, E(K, S(PrivK_A, H_P))))$,

   где V - функция проверки подписи.
   Если ≠, то протокол прерывается.

8. $H_P = H(D(K, E(K, R)) || D(K, E(K, P)) || PubK_A || PubK_B)$,

   Если ≠, то протокол прерывается.

Монолитный криптографический протокол игнорирует способ получения публичного ключа от точки назначения, чтобы таковой оставался встраиваемым и мог внедряться во множество систем, включая одноранговые сети, не имеющие центров сертификации, и тайные каналы связи, имеющие уже установленную сеть по умолчанию. Схема монолитного криптографического протокола на инициирующей стороне показана на *Рисунке 1*.

Протокол способен также игнорировать сетевую идентификацию субъектов информации, замещая её идентификацией криптографической. При таком подходе аутентификация субъектов начинает становиться сингулярной функцией, относящейся лишь и только к асимметричной криптографии, и как следствие, прикладной уровень стека TCP/IP начинает симулятивно заменяться криптографическим слоем по способу обнаружения отправителя и получателя, как это показано на *Рисунках 2, 3*. Из вышеописанного также справедливо следует, что для построения полноценной коммуникационной системы

необходимым является симулятивная замена транспортного и прикладного уровня последующими криптографическими абстракциями. Под транспортным уровнем может пониматься способ передачи сообщений из внешней (анонимной сети) во внутреннюю (локальную), под прикладным — взаимодействие со внутренними сервисами.

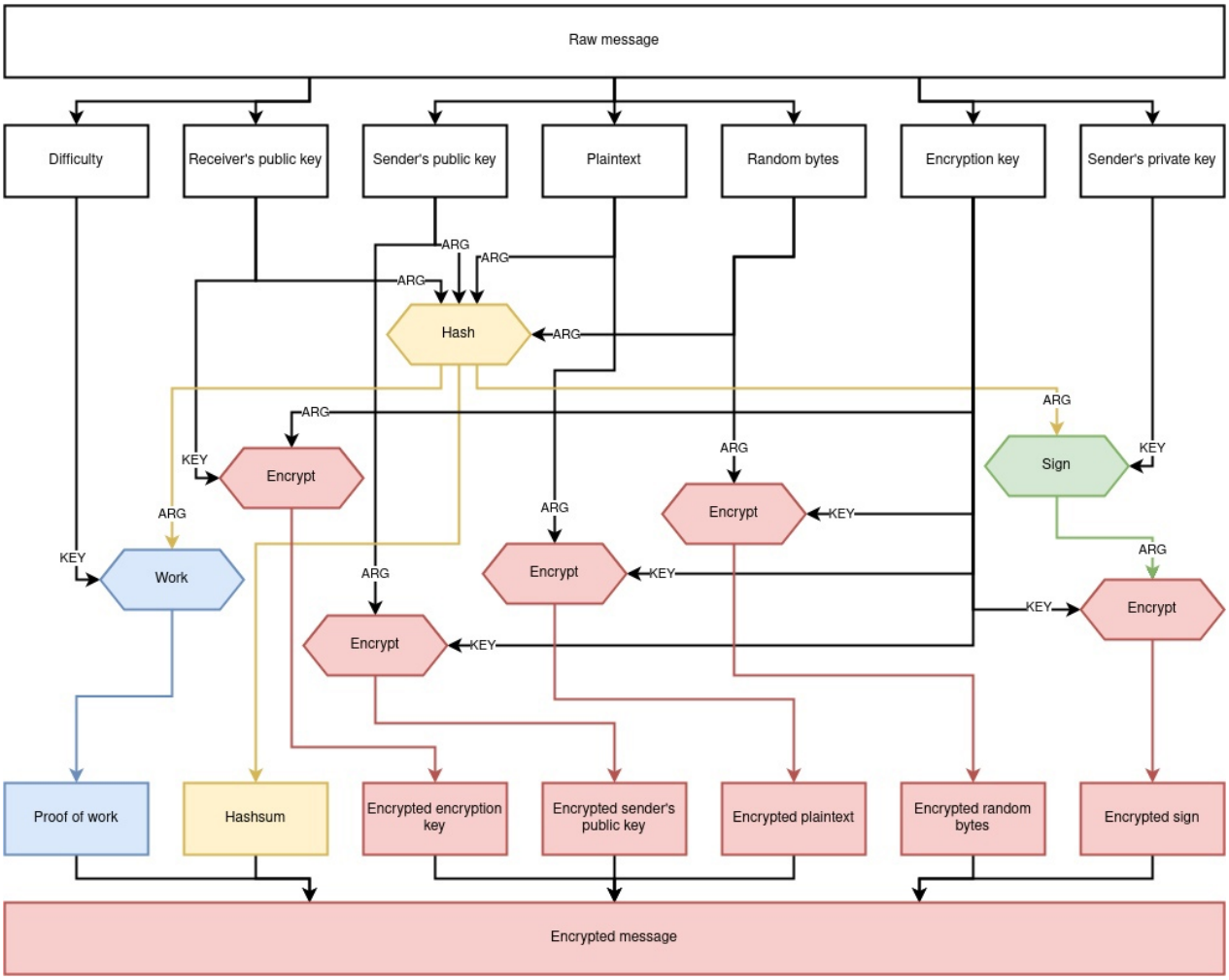

Рисунок 1. Схема монолитного криптографического протокола на инициирующей стороне

Сеанс связи в приведённом протоколе определяется самим пакетом, или иными словами один пакет становится равен одному сеансу за счёт генерации случайного сеансового ключа. Описанный подход приводит к ненадобности сохранения фактического сеанса связи, исключает внешние долговременные связи между субъектами посредством имманентности и абстрагирования объектов, что приводит к невозможности рассекречивания всей информации, даже при компрометации одного или нескольких сеансовых ключей.

Безопасность протокола определяется в большей мере безопасностью асимметричной функции шифрования, т.к. все действия сводятся к расшифрованию сеансового ключа приватным ключом. Если приватный ключ не может расшифровать сеансовый, то это говорит о том факте, что само сообщение было зашифровано другим публичным ключом и потому получатель также есть другой субъект. Функция хеширования необходима для проверки целостности отправленных данных. Функция проверки подписи необходима для аутентификации отправителя. Функция проверки доказательства работы необходима для предотвращения от спама.

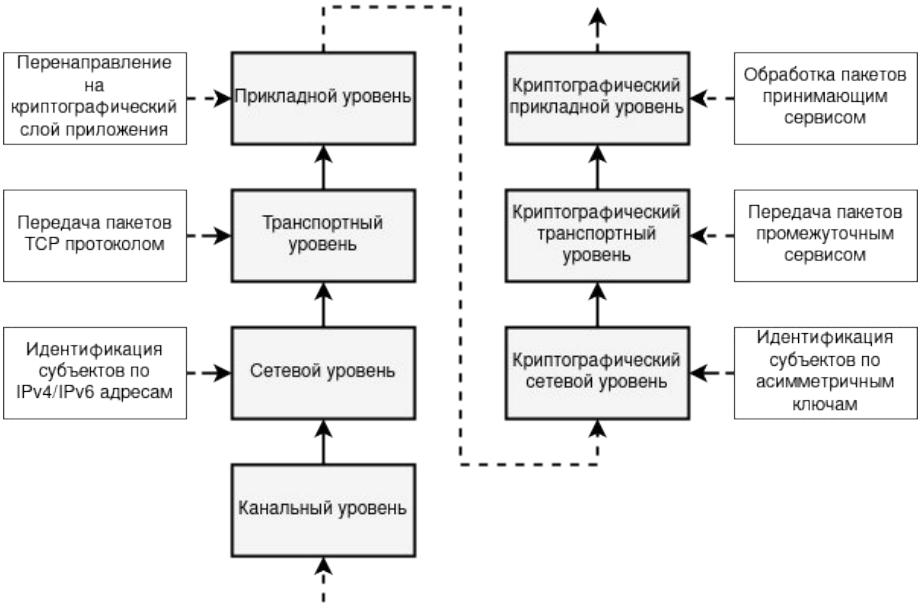

Рисунок 2. Расширение стека протоколов TCP/IP на базе криптографических абстракций

Шифрование подписи сеансовым ключом является необходимым, т.к. взломщик протокола, для определения отправителя (а именно его публичного ключа) может составить список уже известных ему публичных ключей и проверять каждый на правильность подписи. Если проверка приводит к безошибочному результату, то это говорит об обнаружении отправителя.

Шифрование случайного числа (соли) также есть необходимость, потому как, если злоумышленник знает его и субъектов передаваемой информации, то он способен пройтись методом «грубой силы» по словарю часто встречаемых и распространённых текстов для выявления исходного сообщения.

Использование одной и той же пары асимметричных ключей для шифрования и подписания не является уязвимостью, если применяются разные алгоритмы кодирования [3, с.257] или сама структура алгоритма представляет различные способы реализации. Так например, при алгоритме RSA для шифрования может использоваться алгоритм OAEP, а для подписания – PSS. В таком случае не возникает «подводных камней» связанных с возможным чередованием «шифрование-подписание». Тем не менее остаются риски связанные с компрометацией единственной пары ключей, при которой злоумышленник сможет не только расшифровывать все получаемые сообщения, но и подписывать отправляемые [3, с.99][3, с.291]. Но этот критерий также является и относительным плюсом, когда личность субъекта не раздваивается и, как следствие, данный факт не приводит к запутанным ситуациям чистого отправления и скомпрометированного получения (и наоборот).

Протокол пригоден для многих задач, включая передачу сообщений, запросов, файлов, но не пригоден для передачи поточной информации, подобия аудио звонков и видео трансляций, из-за необходимости подписывать и подтверждать работу, на что может уходить продолжительное количество времени. Иными словами, протокол работает с конечным количеством данных, размер которых заведомо известен и обработка которых (то есть, их использование) начинается с момента завершения полной проверки.

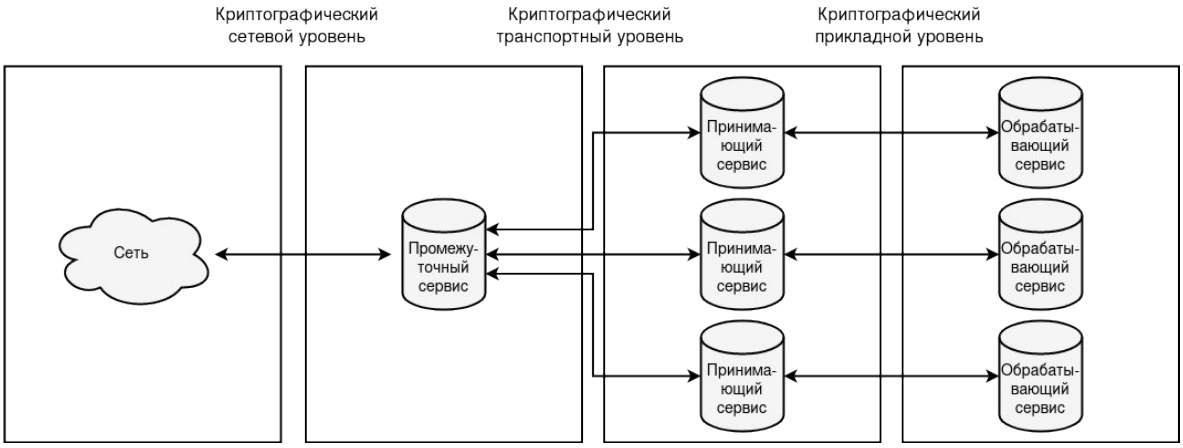

Рисунок 3. Расширенный стек протоколов на примере сервисов в анонимной сети

Недостатком протокола является отсутствие последовательности между несколькими пакетами. Иными словами невозможно определить нумерацию, что в некой степени переводит часть полноценного протокола на логику приложения, как например передача файлов. Это, в свою очередь, обосновывается упрощением протокола, где не требуются хранилище или база данных для хранения последовательности пакетов со стороны каждого входящего объекта. Также в некоторых приложениях последовательность сообщений не критична, как например в электронной почте или мессенджерах, где необходим лишь сам факт уже существующего дубликата (данный момент можно проверять хешем пакета).

Другим недостатком является постоянное применение функции подписания, которая считается одной из наиболее ресурсозатрачиваемой, с практической точки зрения, операцией. При большом количестве поступаемых сообщений, возникнет и необходимость в большом количестве проверок подписания. При этом использование MAC, взамен ЭЦП, является недопустимым, потому как таковая имитовставка создаст буквально поточную связь между субъектами информации (создаст дополнительные связи между субъектами и генерируемым объектом), усложнит протокол и может привести теоретически к более чем одному возможному вектору нападения на протокол.

Протокол можно также расширить и под широковещательную передачу сообщений, где необходимым действием будет являться шифрование одного и того же сеансового ключа пакета несколькими публичными ключами. В итоге, для того, чтобы получить сообщение, получатель должен будет перебрать список зашифрованных экземпляров одного и того же ключа. Расшифровав один из множества ключей, конечный абонент сможет расшифровать и всё сообщение. Недостатком такого подхода является линейное увеличение основной нагрузки на попытки расшифрования сеансового ключа приватным.

Протокол не подвержен timing-атакам (по времени) [4] (не стоит путать данную атаку с timing-атаками по анализу трафика в анонимных сетях), если таковой не участвует в генерации ответа при наступлении стадии прерывания действий принимающей стороной. Иначе, если будет постоянно генерироваться ответ определённым сервисом «принятия всех сообщений», тогда злоумышленник сможет собрать *N*-ое количество пакетов из сети и постепенно отправлять их предполагаемому получателю с постоянным фиксированием времени. Если ответ будет генерироваться дольше среднего значения, то это будет означать повышенную вероятность того, что запрос был отправлен настоящему получателю. Различное время генерации ответа связано с расшифрованием сообщения на уровне протокола, где получатель своевременно проверяет корректность пакета, что может приводить к исключению действий 6, 7 и 8 протокола. Предотвратить timing-атаку можно

сохранением всех принимаемых хеш-значений на стороне сервиса, что приведёт к невозможности повторного использования пакета. Другим решением может являться выставление случайной или статичной задержки при ответе, если возможны случаи отключения определённых участников сети с целью их коммуникационного абстрагирования друг от друга.

Также протокол не подвержен атакам дополнения (padding oracle) [5] при использовании блочных симметричных алгоритмов с режимом шифрования CBC. Невозможность применения данной атаки сводится к вычислению хеш-функции по открытому сообщению. Если злоумышленник будет изменять побайтого данные, стремясь найти правильное значение в шифрованном блоке, таковое действие не будет иметь положительного результата до тех пор, пока все байты не приведут к аналогичному сопоставлению с хеш-значением. По этой причине злоумышленник самостоятельно не сможет выставлять и эффективно проверять корректность промежуточных значений по ответам принимающей стороны.

Протокол способен обеспечивать полиморфизм информации методом установки промежуточных получателей (маршрутизаторов) и созданием транспортировочных пакетов, представленных в форме множественного шифрования. Как только узел сети принимает пакет, он начинает его расшифровывать. Если пакет успешно расшифровывается, но при этом сама расшифрованная версия является шифрованным экземпляром, то это говорит о том, что данный принимающий узел — это промежуточный получатель, целью которого является последующее распространение «расшифрованной» версии пакета по сети. Рекуперация, в совокупности с конечной рекурсией, будет происходить до тех пор, пока не будет расшифрован последний пакет, предполагающий существование истинного получателя, либо до тех пор, пока пакет не распространится по всей сети и не окажется забытым, по причине отсутствия получателя (будь то истинного или промежуточного). Стоит также заметить, что маршрутизаторы при расшифровании пакета могут узнавать криптографический адрес отправителя, именно поэтому стоит отправлять транспортировочные пакеты из-под криптографического псевдо-адреса отправителя.

3. Программная реализация

Монолитный криптографический протокол удобен в программной реализации за счёт своей абстрактности при которой сами алгоритмы шифрования не имеют решающего значения. Таким образом, если один из алгоритмов окажется уязвимым – его можно будет заменить на другой, не изменяя при этом сам протокол.

Пример программного кода[1] [6] для шифрования информации:

```
import (
	"bytes"
)
func Encrypt(sender *PrivateKey, receiver *PublicKey, data []byte) *Package {
	var (
		pubsend		= PublicKeyToBytes(&sender.PublicKey)
		session		= GenerateBytes(N)
		randBytes	= GenerateBytes(N)
	)
```

[1]Программная реализация протокола go-peer [Электронный ресурс]. — Режим доступа: https://github.com/number571/go-peer (дата обращения: 20.03.2022).

```
	hash := HashSum(bytes.Join(
		[][]byte{
			randBytes,
			data,
			pubsend,
			PublicKeyToBytes(receiver),
		},
		[]byte{},
	))

	return &Package{
		Head: HeadPackage{
			Sender:		EncryptS(session, pubsend),
			Session:	EncryptA(receiver, session),
			RandBytes:	EncryptS(session, randBytes),
		},
		Body: BodyPackage{
			Data:	EncryptS(session, data),
			Hash:	hash,
			Sign:	EncryptS(session, Sign(sender, hash)),
			Proof:	ProofOfWork(hash, C),
		},
	}
}
```

Для улучшения эффективности, допустим при передаче файлов, программный код можно изменить так, чтобы снизить количество проверок работы в процессе передачи, но с первоначальным доказательством работы на основе случайной строки (полученной от точки назначения), а потом и с накопленным хеш-значением из *n*-блоков файла, для *i*-ой проверки. Таким образом, минимальный контроль работы будет осуществляться лишь $\lceil M/nN \rceil+1$ раз, где M — размер файла, N — размер одного блока. Если доказательство не поступило или оно является неверным, то нужно считать, что файл был передан с ошибкой и тем самым запросить повреждённый или непроверенный блок заново.

Пример программного кода для расшифрования информации:

```
import (
	"bytes"
)
func Decrypt(receiver *PrivateKey, pack *Package) (*PublicKey, []byte) {
	If len(pack.Body.Hash) != HashSize {
		return nil, nil
	}

	if !ProofIsValid(pack.Body.Hash, C, pack.Body.Proof) {
		return nil, nil
	}

	session := DecryptA(receiver, pack.Head.Session)
	if session == nil {
		return nil, nil
	}

	bpubsend := DecryptS(session, pack.Head.Sender)
	if bpubsend == nil {
		return nil, nil
	}
	pubsend := BytesToPublicKey(bpubsend)
```

```
	if pubsend == nil {
		return nil, nil
	}
	pubsize := PublicKeySize(pubsend)
	if pubsize  != KeySize {
		return nil, nil
	}

	randBytes := DecryptS(session, pack.Head.RandBytes)
	if randBytes == nil {
		return nil, nil
	}

	data := DecryptS(session, pack.Body.Data)
	if data == nil {
		return nil, nil
	}

	check := HashSum(bytes.Join(
		[][]byte{
			randBytes,
			data,
			PublicKeyToBytes(pubsend),
			PublicKeyToBytes(&receiver.PublicKey),
		},
		[]byte{},
	))
	if !bytes.Equal(pack.Body.Hash, check) {
		return nil, nil
	}

	sign := DecryptS(session, pack.Body.Sign)
	if sign == nil {
		return nil, nil
	}

	if !Verify(pubsend, pack.Body.Hash, sign) {
		return nil, nil
	}

	return pubsend, data
}
```

Весь представленный программный код на языке Go представлен только как шаблон, показывающий способ шифрования и расшифрования непосредственно. Проблемой здесь является простота и примитивность анализа сетевого трафика по JSON-формату, что может привести к последующим блокировкам всех сетевых построений на основе данного код, а также игнорирование факта изменения размерности пакета, что может привести к деанонимизации субъектов информации. Необходимым решением предотвращения простоты анализа трафика должно служить вынесение сеансового ключа за пакет JSON-формата, последующее шифрование им пакета и конкатенация зашифрованного пакета с шифрованным сеансовым ключом. Если размер асимметричного ключа заведомо известен, то будет известен и размер зашифрованного сеансового ключа, что не приведёт к каким-либо проблемам расшифрования информации.

Пример программного кода для создания транспортировочного пакета:

```
import (
```

```
	"bytes"
)
func RoutePackage(sender *PrivateKey, receiver *PublicKey, data []byte, route []*PublicKey) *Package {
	var (
		rpack  = Encrypt(sender, receiver, data)
		psender = GenerateKey(N)
	)
	for _, pub := range route {
		rpack = Encrypt(
			psender,
			pub,
			bytes.Join(
				[][]byte{
					ROUTE_MODE,
					SerializePackage(rpack),
				},
				[]byte{},
			),
		)
	}
	return rpack
}
```

Другая проблема заключается в отсутствии каких бы то ни было видимых метаданных (хеш-значения, доказательства работы), которые бы помогли в борьбе со спамом, что в свою очередь является крайне важным критерием для большинства децентрализованных систем. Таким образом, отсутствие метаданных равносильно отсутствию отказоустойчивости, что отсылает на противоречие эквивалентности полностью анализируемого и неподверженного анализу пакетам. Одним из возможных решений данной проблемы может служить использование общепринятного и стандартизированного протокола типа SSL/TLS с целью сокрытия факта использования монолитного протокола.

4. Заключение

Монолитный криптографический протокол является наследником протокола Bitmessage, потому как скрывает внутри шифрованной оболочки сообщения не только маршрутизирующую информацию о её субъектах (отправителе и получателе), но также и всю возможную информацию (криптографическую соль, подписи, версии, расширения), которая так или иначе могла бы выдавать субъектов информации сторонними способами. Монолитный криптографический протокол одновременно является абстрактным, потому как для его функционирования алгоритмы кодирования, шифрования, подписания, хеширования и т.д. не играют решающей роли, и сложно расширяемым, потому как большинство возможных дополнений будет проходить лишь через его более прикладные уровни реализации, но не через модификацию самого протокола. Простота монолитного криптографического протокола становится наиболее выраженным свойством, порождающим его абстрактность, за счёт которой протокол обретает вид класса, соблюдающего основные характеристики монолитности – сокрытие идентификации внутри шифрованной оболочки сообщений.

Список литературы

1. Коваленко, Г. Теория строения скрытых систем [Электронный ресурс]. — Режим доступа: https://github.com/number571/go-peer/blob/master/docs/hidden_systems.pdf (дата обращения: 04.01.2023).
2. Warren, J. Bitmessage: A Peer-to-Peer Message Authentication and Delivery System [Электронный ресурс]. — Режим доступа: https://bitmessage.org/bitmessage.pdf (дата обращения: 31.12.2021).
3. Шнайер, Б., Фергюсон, Н. Практическая криптография / Б. Шнайер, Н. Фергюсон. - М.: Издательский дом «Вильямс, 2005. - 420 с.
4. Атака по времени – сказка или реальная угроза? [Электронный ресурс]. — Режим доступа: https://habr.com/ru/post/217327/ (дата обращения: 08.12.2022).
5. Heaton, R. The Padding Oracle Attack [Электронный ресурс]. — Режим доступа: https://robertheaton.com/2013/07/29/padding-oracle-attack/ (дата обращения: 08.12.2022).
6. Донован, А., Керниган, Б. Язык программирования Go / А.А. Донован, Б.У. Керниган. — М.: ООО «И.Д. Вильямс», 2018. - 432 с.