Hash Concatenation

What does the following code snippet do? Is `concatenatedHash` secure?

```java
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;

public class HashConcatenation {
    private static String hashString(final String input) throws NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] hashBytes = md.digest(input.getBytes(StandardCharsets.UTF_8));
        StringBuilder sb = new StringBuilder();
        for (byte b : hashBytes) {
            sb.append(String.format("%02x", b));
        }
        return sb.toString();
    }

    public static String concatenatedHash(final String input, final int chunkSize) throws NoSuchAlgorithmException {
        List<String> chunks = new ArrayList<>();
        for (int i = 0; i < input.length(); i += chunkSize) {
            int end = Math.min(i + chunkSize, input.length());
            String chunk = input.substring(i, end);
            chunks.add(hashString(chunk));
        }
        return String.join("", chunks);
    }
}
```

Click here to expand...

`concatenatedHash` divides the input string into equal-sized chunks (except for the last chunk, which might be shorter), and then hashes each chunk individually using the `hashString` method. The method finally concatenates all the individual hash strings and returns the resulting string.

The `concatenatedHash` method is insecure. When the input is divided into smaller chunks, the search space for each chunk is much smaller.

It is more secure to hash the entire input in one go, rather than dividing it into chunks and hashing them separately.